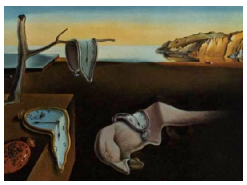

Journal of Statistical Software

February 2008, Volume 24, Issue 9. *http://www.jstatsoft.org/*

A statnet Tutorial

Steven M. Goodreau
University of Washington

Mark S. Handcock
University of Washington

David R. Hunter
Penn State University

Carter T. Butts
University of California, Irvine

Martina Morris
University of Washington

Abstract

The **statnet** suite of R packages contains a wide range of functionality for the statistical analysis of social networks, including the implementation of exponential-family random graph (ERG) models. In this paper we illustrate some of the functionality of **statnet** through a tutorial analysis of a friendship network of 1,461 adolescents.

Keywords: relational data, social networks, exponential-family random graph models, **statnet**, R.

1. Introduction

statnet comprises a suite of R (R Development Core Team 2007) packages for the analysis of social networks and other relational data. Previous papers in this issue describe in detail the functionality for most of these packages, and the underlying statistical theory behind it. In this paper we illustrate some of the functionality of three core packages within the **statnet** suite—**network**, **sna**, and **ergm**—through a tutorial-based analysis of a friendship network with 1,461 vertices.

Readers of a more methodical nature may wish to examine the rest of the papers in this issue first. For those who do wish to tackle the tutorial early on, and who do not have much familiarity with statistical network models, we suggest that you begin by reading at least the introduction to this issue (Handcock, Hunter, Butts, Goodreau, and Morris 2008). In the tutorial, we assume a basic familiarity with R (including how to obtain further help or documentation for any of the commands we use) and with a variety of concepts and terminology from social network analysis. For newcomers, some of the latter may be understandable from context with the aid of a social network reference text, although those sections pertain-

ing to exponential-family random graph (ERG) models will likely require additional reading (Wasserman and Pattison 1996; Robins, Pattison, Kalish, and Lusher 2007a; Hunter, Handcock, Butts, Goodreau, and Morris 2008b).

The **statnet** suite may be used to analyze a variety of types of network data from many different substantive domains, including directed and undirected, and one-mode and two-mode (bipartite) networks. Some functionality also exists in the current version for handling dynamic networks and networks with missing data, both of which are undergoing continual development. However, for clarity, this tutorial will focus on a single dataset: a static, undirected, one-mode friendship network of 1,461 vertices. It is derived from data from the National Longitudinal Study of Adolescent Health, or Add Health (Udry 2003; Harris, Florey, Tabor, Bearman, Jones, and Udry 2003) by a process described in the Appendix. We call this dataset "Faux Magnolia High School"—Magnolia, because the schools on which it is based are large and located in the Southern US, and Faux because it is a model-based simulation from the original data, since the latter are subject to confidentiality restrictions. The data and analyses conducted in this tutorial are purposefully chosen to resemble those previously conducted by our group (Goodreau 2007; Hunter, Goodreau, and Handcock 2008a; Goodreau, Kitts, and Morris 2008).

Throughout the tutorial, R input is represented by italicized typewriter font beginning with the `R>` prompt, or `+` prompt if it is a continuation of a previous line. Be sure to remove these symbols if copying and pasting commands into R. For convenience, all commands are also available in a standalone text file `v24i09.R`. Output from R is represented by regular typewriter font without these prompts, and we sometimes edit lengthy output and indicate deleted text using an ellipsis (`...`).

2. Obtaining the statnet suite of packages

The packages in the **statnet** suite are connected via the meta-package **statnet**, which is little more than a wrapper for the other packages. To install and attach all packages, simply open R, and type:

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

Some users may only wish to obtain a subset of these packages. For this tutorial, only **network**, **sna** and **ergm** are required. One may install and attach each individually, although a shortcut is simply to install and attach **ergm** (which depends on **network**) and **sna**. Each of the individual packages mentioned here, along with the **statnet** package itself, is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/`. In addition, more information about the **statnet** packages is available on the Web at `http://statnetproject.org/`.

3. License and citation information

All **statnet** packages are free and open-source and are released under GPL (General Public License). The packages **network** and **sna** are released under GPL-2, while **ergm** is released

under GPL-3 with attribution requirements for the source code and documentation. To obtain license information for all packages, simply type `license.statnet()`, or go to http://statnetproject.org/attribution/.

Please cite the **statnet** packages when you use them for research that is published or otherwise publicly distributed. Citation information is provided on our Web site at http://statnetproject.org/citation.shtml, and can be obtained by typing `citation("statnet")`, or for any of the constituent packages by typing `citation("<name.of.package>")`.

4. Network exploration

In this section, we explore methods for querying our data. We do not cover methods for importing, converting or manipulating network data; for information on these functionalities, please see Butts (2008a).

To begin, make sure that the necessary packages are loaded into your current session (with `library("statnet")` or with both `library("ergm")` and `library("sna")`) and then load the dataset we will be analyzing, found in the **ergm** package:

```
R> data("faux.magnolia.high")
```

A shorter name will save us much typing effort:

```
R> fmh <- faux.magnolia.high
```

`fmh` is an object of class `network`; to see what information is contained in a `network` object, type:

```
R> fmh

Network attributes:
   vertices = 1461
   directed = FALSE
   hyper = FALSE
   loops = FALSE
   multiple = FALSE
   bipartite = FALSE
   total edges= 974
...
```

As this output is extensive (and includes the entire network edge list), it is often more useful to look at an abridged version of this information, which may be obtained by typing `summary(fmh)`. Among other things, we see that the network has 1,461 vertices and 974 edges.

Before we begin modeling these data statistically, it is good to have a general handle on their nature; perhaps the easiest way to do so for network data is by visualizing them. Since this is a large, sparse network, it is probably best to leave the isolates out:

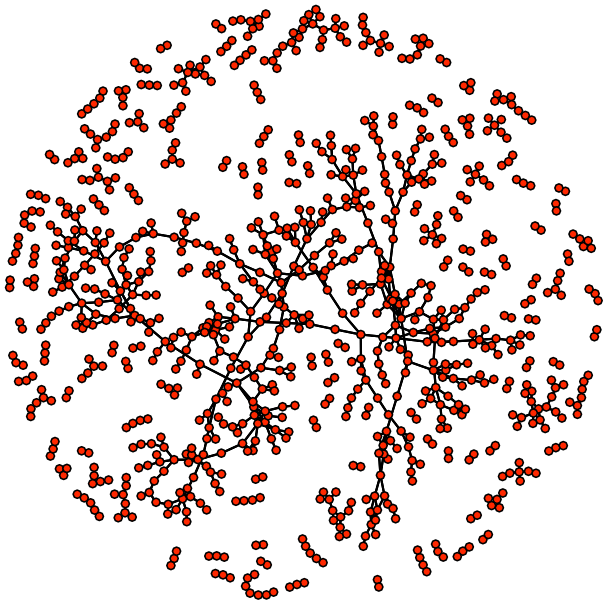

Figure 1: Faux Magnolia High, without isolates.

```
R> plot(fmh, displayisolates = FALSE, vertex.cex = 0.7)
```

Note that this command might take upwards of a minute or two, depending on the speed of your computer. There appears to be one large component and a smattering of many very small components (Figure 1), not to mention all of the components of size 1 that we removed from the visualization. To get a count of the component size distribution:

```
R> table(component.dist(fmh)$csize)
```

```
  1   2   3   4   5   6   7   8   9  11  12  19  23 439
524  64  29  11  12   4   7   4   1   1   1   1   1   1
```

From the output, we see that there are 524 isolates (which were not included in the visualization), one large component of 439 vertices, and many components in between.

Let us look at an excerpt of the output given by the network summary:

```
R> summary(fmh)
```

```
...
 Vertex attributes:

vertex.names:
   Length     Class      Mode
     1461 character character
```

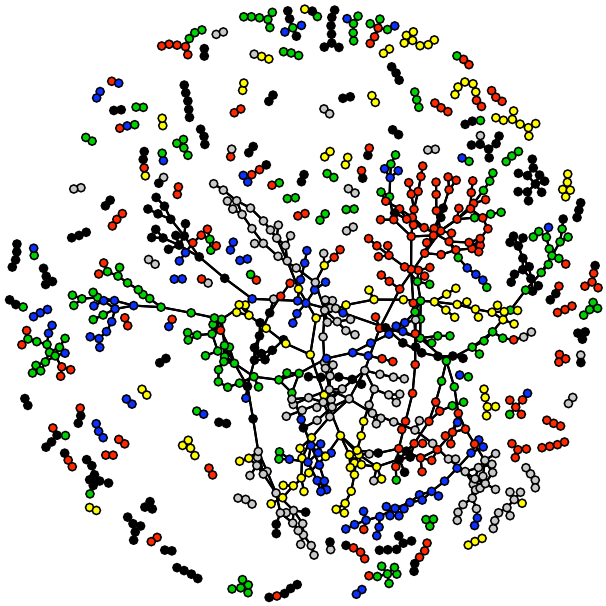

Figure 2: Faux Magnolia High, without isolates, colored by grade.

```
Race:
Asian Black  Hisp NatAm Other White
   48   261    68    24     7  1053

Grade:
  7   8   9  10  11  12
185 210 317 299 257 193

Sex:
  F   M
768 693
```

There are four different vertex attributes associated with the vertices in this network: "vertex.names", "Race", "Sex", and "Grade." Grade is likely to be a strong determinant of social relations, so let us visualize the network with vertices colored by their grade:

```
R> plot(fmh, displayisolates = FALSE, vertex.col = "Grade", vertex.cex = 0.7)
```

Clearly there is a strong tendency for friendships to form within grade (Figure 2), although at the same time the grades themselves do not appear to be cohesive units; rather each grade seems to consist of a number of subgroups which link together. One might choose to explore other visualization options by typing `help(plot.network)`.

Network modelers are frequently interested in the degree distribution:

```
R> fmh.degreedist <- table(degree(fmh, cmode = "indegree"))
R> fmh.degreedist
```

```
  0   1   2   3   4   5   6   7   8
524 403 271 128  85  30  13   5   2
```

The `cmode` argument tells the `degree` command that since these are undirected data, we do not want to count both the upper and lower triangles of the sociomatrix, but rather only one. We could have chosen `"outdegree"` instead with identical results; leaving the argument off would yield numbers in the upper row double those seen above.

The **sna** package provides individual functions for calculating a wide array of classic measures on networks, such as the previous command, `degree`. At the same time, the **ergm** package provides means for calculating any of the network statistics available for ERG modeling, all through the single command `summary`. For instance:

```
R> summary(fmh ~ degree(0:8))
```

yields the same numerical results as the previous command from **sna**:

```
degree0 degree1 degree2 degree3 degree4 degree5 degree6 degree7 degree8
    524     403     271     128      85      30      13       5       2
```

Additional summary measures are available in both **sna** and **ergm**. Although there is some overlap in functionality, several functions remain unique within a package, so checking both for particular measures of interest may be helpful. A list of network statistics available for ERG models can be found in this issue (Morris, Handcock, and Hunter 2008); and for **sna**, in the package's help files, via:

```
R> help(package = "sna")
```

One might also wish to compare degree distributions by sex:

```
R> summary(fmh ~ degree(0:8, "Sex"))
```

```
deg0.SexF deg1.SexF deg2.SexF deg3.SexF deg4.SexF deg5.SexF deg6.SexF
      226       226       160        80        44        18         7
...
```

We leave it to the interested reader to split this out by sex, re-express as proportions and plot. Instead, we turn to another network feature that is commonly of interest, the count of triangles:

```
R> summary(fmh ~ triangle)
```

```
triangle
     169
```

Instinct tells us that 169 triangles is many more than would be expected by chance in a network of 1,461 actors and 974 edges. We will revisit this in a more rigorous fashion below. Note that one can obtain more than one statistic at a time using this function; simply separate them with the plus symbol (standard syntax within R):

```
R> summary(fmh ~ edges + triangle)

   edges triangle
     974      169
```

When we visualized the network we saw a preponderance of within-grade edges. Let us display the mixing matrix by grade (i.e., the count of relationships cross-tabulated by the grades of the two actors involved):

```
R> mixingmatrix(fmh, "Grade")

Note:  Marginal totals can be misleading
 for undirected mixing matrices.
     7   8   9  10  11  12
7  110  11   3   3   0   0
8   11 165   9   7   0   2
9    3   9 152  24  10   4
10   3   7  24 151  38  11
11   0   0  10  38 152  32
12   0   2   4  11  32  90
```

Clearly most of the contacts are concentrated on the diagonal; you may wish to examine the mixing matrices for `Race` and `Sex` as well. To see a vector of attribute values for all of the actors for one of these attributes, or a table of the same, use the `get.vertex.attribute` command, or its shortcut, `%v%`:

```
R> gr <- fmh %v% "Grade"
R> table(gr)

  7   8   9  10  11  12
185 210 317 299 257 193
```

For more information on importing, exporting and manipulating `network` objects, see Butts (2008a) or the documentation for the **network** package. For more information on performing classical network analysis, see Butts (2008b) or the documentation for the **sna** package.

Meanwhile, saving your work is always wise:

```
R> save.image()
```

5. Fitting an ERG model

The **ergm** package allows the user to fit exponential-family random graph (ERG) models to network data sets. These models, also known as p* (p-star), are described in great detail in earlier papers in this issue (Handcock *et al.* 2008; Hunter *et al.* 2008b), and elsewhere in the literature (Wasserman and Pattison 1996; Snijders, Pattison, Robins, and Handcock 2006;

Robins and Morris 2007). Very briefly, the model class formulates the probability of observing a set of network edges (and non-edges) as

$$\mathsf{P}(\mathbf{Y} = \mathbf{y}|\mathbf{X}) = \exp[\boldsymbol{\theta}^T \mathbf{g}(\mathbf{y}, \mathbf{X})]/\kappa(\boldsymbol{\theta}),$$

where $\mathbf{Y}$ is the (random) set of relations (edges and non-edges) in a network, $\mathbf{y}$ is a particular given set of relations, $\mathbf{X}$ is a matrix of attributes for the vertices in that network, $\mathbf{g}(\mathbf{y}, \mathbf{X})$ is a vector of network statistics, $\boldsymbol{\theta}$ is the vector of coefficients, and $\kappa(\boldsymbol{\theta})$ is a normalizing constant. Equivalently, the model states that the log-odds that any given edge will exist given the current state of the rest of the network is

$$\text{logit}(Y_{ij} = 1) = \boldsymbol{\theta}^T \boldsymbol{\delta}[\mathbf{g}(\mathbf{y}, \mathbf{X})]_{ij}\,,$$

where Y_{ij} is an actor pair in $\mathbf{Y}$ (which is ordered if $\mathbf{Y}$ is directed and unordered if not), and $\boldsymbol{\delta}[\mathbf{g}(\mathbf{y}, \mathbf{X})]_{ij}$ is the change in $\mathbf{g}(\mathbf{y}, \mathbf{X})$ when the value of y_{ij} is toggled from 0 to 1.

Let us begin with the simplest model of interest, a single-parameter model that posits an equal probability for all edges in the network. This model is known in different branches of network science as the Bernoulli model or the Erdös-Rényi model, and it is a natural null model from which to proceed. In the ERG modeling framework, this corresponds to a model with a $\mathbf{g}(\mathbf{y}, \mathbf{X})$ vector of statistics that contains only a single element, the number of edges in the network.

```
R> model1 <- ergm(fmh ~ edges)
R> summary(model1)

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

Formula:   fmh ~ edges

Newton-Raphson iterations:  8

Maximum Likelihood Results:
      Estimate Std. Error MCMC s.e. p-value
edges -6.99760    0.03205        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: 1478525  on 1066530  degrees of freedom
 Residual Deviance:   15580  on 1066529  degrees of freedom
          Deviance: 1462944  on       1  degrees of freedom

AIC: 15582    BIC: 15594
```

The coefficient estimate tell us that if this network had been generated by the posited model, then the log-odds of a tie should equal -6.998 $\times$ $\delta(g(\mathbf{y}, \mathbf{X}))_{ij}$, which would equal -6.998 for all edges, since the addition of any edge to the network changes $g(\mathbf{y}, \mathbf{X})$, the total number of edges in the network, by 1.

The probability that corresponds to this log-odds is

$$\exp(-6.998)/(1 + \exp(-6.998)) = 0.000913$$

The `summary(fmh)` command reveals that the density of the network, that is, the fraction of possible edges that are realized, is indeed 0.000913. This model is sufficiently trivial that the coefficient could have been obtained analytically; this will not remain the case for long. In addition, the model fit not only tells us the point estimate for the log-odds of a tie, but the standard error of those log-odds as well, thus providing additional information about our level of certainty in the point estimate.

Other components of the output include information about the amount of deviance explained by the model, two standard measures of model fit based on the likelihood (AIC and BIC), and the MCMC standard error (i.e. the additional uncertainty in the coefficient estimates induced by the MCMC estimation procedure). Since this model exhibits dyadic independence, MCMC estimation is not necessary, and the MCMC standard error does not apply. For more information on all of these concepts, see the preceding papers in this issue, especially Hunter *et al.* (2008b).

There is much more to an `ergm` object than that which is shown in the summary:

```
R> names(model1)
```

```
 [1] "coef"          "sample"              "iterations"      "mle.lik"
 [5] "MCMCtheta"     "loglikelihoodratio" "gradient"        "hessian"
 [9] "covar"         "samplesize"          "failure"         "mc.se"
[13] "glm"           "glm.null"            "null.deviance"   "aic"
[17] "theta1"        "offset"              "drop"            "network"
[21] "newnetwork"    "formula"             "constraints"     "prop.weights"
```

The user may find out more about each of these components by examining the "Value" section under `help(ergm)`. Those which one is typically most interested in extracting for analysis and model comparison would include the coefficients and the likelihood:

```
R> model1$coef
```

```
    edges
-6.997596
```

```
R> model1$mle.lik
```

```
[1] -7790.104
```

Since we have all lived through adolescence, we know full well that the network of high school friendships is not simply random. How can we explore alternative hypotheses about the structure that exists and the underlying processes that generated it? This is precisely the strength of the ERG modeling framework.

A good model with which to begin is one that proposes a tendency for assortative mixing—that is, a greater probability of individuals forming edges with others of the same race, sex, or grade as themselves. This model is a convenient one with which to extend our investigations since it exhibits dyadic independence. As explained in Hunter *et al.* (2008b), such models are generally easier to fit than others, and the fitting process itself is not usually affected if the model is inappropriate for the data. In this case, it is also a very reasonable hypothesis to propose.

Let us test for a tendency towards assortative mixing that is uniform within each attribute class—i.e. there is the same tendency for within-race edges (for example), regardless of which race one is talking about. This model can be fit using the `nodematch` term, with the default argument of `diff = FALSE`:

```
R> model2 <- ergm(fmh ~ edges + nodematch("Grade") + nodematch("Race") +
+    nodematch("Sex"))
R> summary(model2)

...
                 Estimate Std. Error MCMC s.e. p-value
edges           -10.01277    0.11526        NA  <1e-04 ***
nodematch.Grade   3.23105    0.08788        NA  <1e-04 ***
nodematch.Race    1.19646    0.08147        NA  <1e-04 ***
nodematch.Sex     0.88438    0.07057        NA  <1e-04 ***
...

R> model2$mle.lik

[1] -6528.714
```

We see that all four terms are significant; and we also see a dramatic increase in model likelihood relative to model 1. One could conduct a formal likelihood ratio test between models 1 and 2 if one wished. Note that in practice, if one is including `nodematch` terms in a model, one would typically also include `nodefactor` terms for the same attributes (since the latter capture "main effects" of attributes whereas the former can be thought of as capturing "interaction" or "second-order effects" of attributes).

One can interpret the coefficients of this model in terms of the log-odds of different types of ties: the log-odds of a tie that is completely heterogeneous (the two members differ from each other in race, sex, and grade) is -10.01; the log-odds of a tie that is homogeneous by race only is -8.82 ($= -10.01 + 1.20$, with rounding error); one that is homogeneous in all three attributes is -4.70 ($= -10.01 + 3.23 + 1.20 + 0.88$), etc.

Let us delve a little deeper to see what this model does and does not capture about the structure in our original data. To do this, we use the model fit that we have just generated to simulate new networks at random, and consider how these are similar to or different from

our data. Because this and many of the following commands rely on pseudo-random number generation, we use the `seed` argument throughout the article to seed the random number generator. This makes the output exactly reproducible, though we have noticed that for certain commands, the output sometimes differs on certain platforms (we used R version 2.6.2 for Windows to obtain this output). Simulating networks from an `ergm` object is done via the `simulate` command:

```
R> sim2 <- simulate(model2, burnin = 1e+6, verbose = TRUE, seed = 9)

Starting 1 MCMC iteration of 1e+06 steps.
   TNT sampler accepted 33.144% of 1000000 proposed steps.
```

Here we have changed the burn-in (the number of steps in the simulation chain before the simulated network is drawn) from the default of 1000. When simulating a single network this allows the chain a chance to move away from the starting network so that the output is approximately independent of initial conditions. Using `verbose = TRUE` allows the user to see what percentage of the million proposals were accepted, and thus to check how far the simulation was capable of moving. For the above example, approximately 33%, or 330,000 proposals, were accepted, allowing plenty of opportunity for a network of only 974 edges to move away from its initial conditions. (The interested reader might wish to try the same `simulate` command, but accept the default burn-in of 1000, and see how the resulting network compares to the data).

The default for the `simulate` command is to simulate a single network, and thus the object returned from the above command is of class `network`. You may examine the mixing matrices for the attributes in this network and in the data to see how they compare:

```
R> mixingmatrix(sim2, "Race")

...
      Asian Black Hisp NatAm Other White
Asian     0     6    1     3     0    16
Black     6    39    9     3     2   116
Hisp      1     9    4     1     0    35
NatAm     3     3    1     0     0     8
Other     0     2    0     0     0     4
White    16   116   35     8     4   708

R> mixingmatrix(fmh, "Race")

...
      Asian Black Hisp NatAm Other White
Asian     7     4    0     1     0    35
Black     4    85    9     3     0    57
Hisp      0     9    1     0     0    48
NatAm     1     3    0     3     0    25
Other     0     0    0     0     0     5
White    35    57   48    25     5   691
```

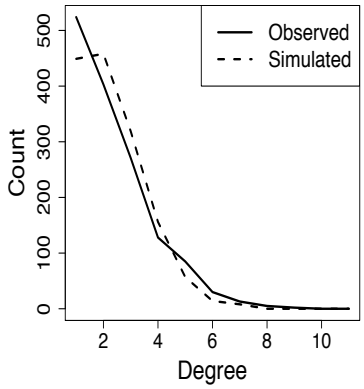

Figure 3: Degree distribution, data vs. model 2.

How are the matrices above similar to, and different from, one another?

Let us now compare our model fit to our data on another feature of great interest to many network modelers: the degree distribution.

```
R> plot(summary(fmh ~ degree(0:10)), type = "l", lty = 1, lwd = 2,
+    xlab = "Degree", ylab = "Count")
R> lines(summary(sim2 ~ degree(0:10)), lty = 2, lwd = 3)
R> legend("topright", legend = c("Observed", "Simulated"), lwd = 3,
+    lty = 1:2)
```

You should get the plot shown in Figure 3. This relatively simple model generates a network that matches the observed network in the upper tail but that is inconsistent toward the lower end of the distribution, most notably in the relative proportion of vertices with degree 0 and 1 (index 1 and 2, respectively). One might wish to conduct a more rigorous test to determine whether these differences are significant or not; we tackle this topic Section 8 below.

How do these two networks compare with regard to triangles?

```
R> c(fmh = summary(fmh ~ triangle), sim2 = summary(sim2 ~ triangle))
```

```
 fmh.triangle sim2.triangle
          169             2
```

Clearly, this model does not capture the process that generated the observed level of triangles. Perhaps, then, one might wish to fit a simple triangle model, with only edge and triangle terms. Or, one might wish to add a triangle term to model 2, considering both triangles and assortative mixing together, since it is fairly clear that assortative mixing is at work. Or, one might wish to explore the classic homogeneous Markov model (edges, triangle, k-star terms). Unfortunately, all of these models are *degenerate* for this network, in the sense used by Handcock (2003a,b). That is, they are such poor descriptions of the underlying process—the observed network is so unlikely even under the maximum likelihood coefficients—that the model cannot even be properly estimated.

6. Identifying model degeneracy

To observe how exactly degeneracy plays out in practice and how to diagnose it with your own model estimation, let us work through one example. Let us first fit the triangle model with all of the defaults for `ergm` (and a value of `seed` to make the output reproducible, as well as a flag to return some additional degeneracy diagnostics):

```
R> model3 <- ergm(fmh ~ edges + triangle, seed = 99)

Iteration 1 of at most 3: the log-likelihood improved by 1.373
Iteration 2 of at most 3: the log-likelihood improved by 0.9386
Iteration 3 of at most 3: the log-likelihood improved by 2.837
...
```

This model is a dyadic dependence model, so the fitting algorithm draws on MCMC and is thus stochastic.

Let us see what the initial and final coefficients look like:

```
R> model3

Newton-Raphson iterations:  29
MCMC sample of size 10000 based on:
   edges  triangle
  -7.254     4.558

Monte Carlo MLE Coefficients:
   edges  triangle
  -7.310     4.584
```

From this output, things might seem relatively good: the model yielded coefficient estimates that were slightly different from the starting values, but not wildly so. Moreover, the improvements in the log-likelihood values are reasonably low numbers. (A number approaching 20.0, the point at which the algorithm cuts off further estimation, usually indicates problems with convergence). It is good if the successive changes in log-likelihood get small (e.g., less than 1) and are generally decreasing. So we should look more closely. Let us do so by examining the diagnostics for the MCMC model fitting process:

```
R> pdf("model3diagnostics.pdf")
R> mcmc.diagnostics(model3)
R> dev.off()
```

This requires the **coda** package (Plummer, Best, Cowles, and Vines 2007) to be installed from CRAN. The `mcmc.diagnostics` function creates a PDF file in your working directory that should resemble Figure 4 (along with some text output, which we omit here). The plots tell the user what is happening to the model statistics during the last iteration of the MCMC estimation procedure, in this case iteration 3 since the default for `maxit` in `ergm` is 3. The left-hand plots represent the chain as one time series for each model statistic, while the right-hand side summarizes this chain in a histogram. Both are normalized to the observed data,

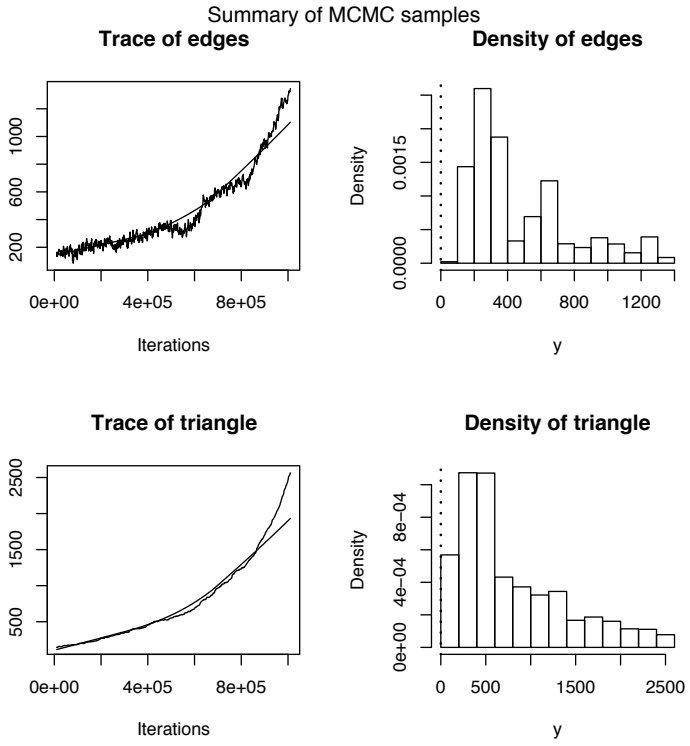

Figure 4: MCMC diagnostics, model 3.

represented by 0. In a converged model, these statistics will vary stochastically around the mean, but will not trend steadily away from the mean, as they do here. This tells us that our initial estimates for the coefficient values generate networks with many more triangles than actually observed, which will make maximum likelihood estimation impossible. This is good evidence that one has a degenerate model.

How else might one detect this? Some of the feedback given in verbose mode also helps (the output below is produced during the third of three iterations):

```
R> model3 <- ergm(fmh ~ edges + triangle, verbose = TRUE, seed = 8)
```

```
...
Summary of simulation, relative to observed network:
                   edges   triangle
Min.             78.00000  144.00000
1st Qu.         235.00000  314.00000
Median          337.00000  568.00000
Mean            468.70000  784.10000
3rd Qu.         640.00000 1132.00000
Max.           1355.00000 2566.00000
std. err.         3.07605    6.02356
std. t units.   152.37070  130.17197
p.value dev.      0.00000    0.00000
```

```
std. dev.      307.60495  602.35625
...
```

This tells us that in this particular simulation the mean for the edge statistic was off by an average of 468.7, and the triangle statistic by 784.1. Even the smallest simulated edge and triangle statistics were 78 and 144 larger than the observed statistics, respectively. Given the high instability in this model fit, repetitions of this command may give results that look considerably different. These are all further evidence of degeneracy. Another means for diagnosis that uses the same principle is simply to simulate from the final coefficient estimates and compare the simulated network to the observed for the statistics in the model.

On the flip side, one must be careful not to assume model degeneracy too quickly, either. The problems with convergence just described can in some cases arise not because the model is a degenerate one, but simply because the Markov chain has not been allowed to run long enough to cover the sample space sufficiently. This is especially true if the starting values for the estimation are very far from the true MLE. Before one settles on a diagnosis of degeneracy, it is best to explore some of the following options. For this particular model, none of these will help greatly, but it is still an informative exercise.

One straightforward approach is to increase the MCMC sample size, or interval, or both:

```
R> model3.take2 <- ergm(fmh ~ edges + triangle, MCMCsamplesize = 1e+5,
+    interval = 1000, verbose = TRUE, seed = 88)
```

The output should contain the following comment:

```
The network has more than 50000 edges, and the model is likely to be
degenerate.
```

which is a very clear sign that model convergence has failed.

Another approach is to decrease the step length, increase the maximum iterations through the simulation-estimation cycle, or both:

```
R> model3.take3 <- ergm(fmh ~ edges + triangle, maxit = 25, seed = 888,
+    control = control.ergm(steplength = 0.25), verbose = TRUE)
```

In this case, the model eventually returns a similar warning as for `model3.take2`, starting at iteration 4. Note that in the first three iterations, the algorithm seems to stabilize at a set of coefficient estimates around -7.3 and 4.5. However, this stability is again a ruse: examining the MCMC diagnostics and/or simulating from these intermediate coefficient values will reveal that this model has indeed failed to converge.

A final option is to consider a different proposal type for the MCMC, which may introduce some constraints on the estimation process. The default MCMC proposal in **ergm** has no constraints, but does weight the selection of ties ($y_{ij} = 1$) and non-ties ($y_{ij} = 0$) so as to ensure that each class is selected with overall probability 0.5 regardless of network density. In statnet, this proposal is called `TNT` (i.e. "tie/no-tie"). However, one might choose a proposal type that constrains the sample space, e.g. one that fixes the total number of edges or the nodal degrees. These can be selected by adding the argument `constraints = ~ edges` or `constraints = ~ degrees` to the `ergm` command, respectively. Doing so should be based on

a theoretical belief that this constraint appropriately captures the stochastic process behind the network; if not, it may simply mask degeneracy rather than overcoming it. Other options for constraints or weights for the MCMC proposals are described in Morris *et al.* (2008) or can be found in the help for the `ergm` command.

7. Non-degenerate dyadic dependence models

Let us explore clustering using one of the more recently proposed statistics that is closely related to the triangle, but has proven to be more robust to degeneracy in practice. This is the GWESP (geometrically weighted edgewise shared partner) statistic; For more information on it, and the equivalent "alternating k-triangle" statistic, see Snijders *et al.* (2006); Robins, Snijders, Wang, Handcock, and Pattison (2007b) or Hunter (2007). Since we are fairly confident that the assortative mixing processes are real, we shall add this new term to model 2. The GWESP statistic has one scaling parameter (α) as well as the usual coefficient. Although **ergm** contains functionality to fit the MLE of both parameters simultaneously, this generally requires much longer computation time to converge. For this tutorial we will keep things simpler and explore a number of models with fixed α.

When the scaling parameter equals 0, the statistic is equivalent to the number of edges that are in at least one triangle; when it approaches infinity, the statistic approaches three times the number of triangles. The lower the value of α, the less likely the model is to be degenerate, so we may wish to start close to zero and move up, looking to see how well we do in matching the count of triangles:

```
R> model4.take1 <- ergm(fmh ~ edges + nodematch("Grade") +
+    nodematch("Race") + nodematch("Sex") + gwesp(0, fixed = TRUE),
+    MCMCsamplesize = 1e+5, maxit = 15, verbose = TRUE,
+    control = control.ergm(steplength = 0.25), seed = 123)
```

Note that these and subsequent model estimation runs may take upwards of 15–20 minutes each. With `verbose` set to `TRUE`, you should receive occasional feedback from R about the progress of the algorithm, although at other times the program may or may not be available depending on your operating system.

How well did the fitting algorithm do? How do the diagnostic plots look? If you simulate a single network from this model, how many edges and triangles does it contain?

Interpreting the coefficients for the GWESP model gets much more complex than the previous dyadic independence models:

```
R> model4.take1$coef
```

```
      edges nodematch.Grade  nodematch.Race   nodematch.Sex   gwesp.fixed.0
 -9.8297890       2.7904821       0.9511466       0.7877837       1.8031250
```

As an example, imagine a pair of actors who differ from each other on all attributes. If they have no friends in common, then the log-odds of them becoming friends is -9.83. If they have *any* positive number of friends in common, and each of them is in at least one other triangle with each of those friends, then the log-odds of them becoming friends rises to -8.03

($= -9.83 + 1.80$). If α had had a positive value (unlike this case, where it equals 0), then the log-odds of them becoming friends would have increased monotonically (but sub-linearly) with the number of friends they had in common. If some of the two actors' common friends were not already in other triangles with each of them, then the log-odds rises even more: in the case of $\alpha = 0$, by an additional 1.80 for each edge that is not in any triangle but that enters one when the two actors become friends.

Now try using the GWESP term with some different values for the α parameter, such as 0.1 and 0.2:

```
R> model4.take2 <- ergm(fmh ~ edges + nodematch("Grade") +
+    nodematch("Race") + nodematch("Sex") + gwesp(0.1, fixed = TRUE),
+    MCMCsamplesize = 1e+5, maxit = 15, verbose = TRUE,
+    control = control.ergm(steplength = 0.25), seed = 123)

R> model4.take3 <- ergm(fmh ~ edges + nodematch("Grade") +
+    nodematch("Race") + nodematch("Sex") + gwesp(0.2, fixed = TRUE),
+    MCMCsamplesize = 1e+5, maxit = 15, verbose = TRUE,
+    control = control.ergm(steplength = 0.25), seed = 123)
```

Notice that the only difference among the last three models is the value of the α argument to the `gwesp` term. Try repeating the above command for progressively higher values of α, until the model loglikelihood ceases to improve or you get bored.

You can compare the approximate maximized values of the loglikelihood function for each of the last three models via:

```
R> c(model4.take1$mle.lik, model4.take2$mle.lik, model4.take3$mle.lik)

[1] -5604.181 -5549.739 -5502.275
```

Or, you may wish to simulate one or more networks from each and compare their triangle counts to the original.

You should also examine the estimated coefficients for the three different models by typing `summary(model4.take1)`, etc. Note that no matter what value is used for α within this range, all of the terms are significant and the coefficients do not change all that much. We select the version with an α of 0.2 to continue; you should feel free to pick whichever you wish:

```
R> model4 <- model4.take3
R> model4$coef

      edges nodematch.Grade  nodematch.Race   nodematch.Sex gwesp.fixed.0.2
 -9.7915376       2.7557927       0.9184486       0.7664999       1.8150806
```

In Table 1, we compare our three non-degenerate models on likelihood and coefficients. We see that model 4 does indeed improve the likelihood of the model considerably. Note also the difference between the model 4 coefficients and those in model 2; the coefficients on all of the match terms decline in the face of the new triangle statistic, while the edges term has

	model 1	model 2	model 4
log likelihood	−7790.10	−6528.71	−5502.28
edges	−7.00	−10.01	−9.79
nodematch.Grade		3.23	2.76
nodematch.Race		1.20	0.92
nodematch.Sex		0.88	0.77
gwesp.fixed.0.2			1.82

Table 1: Model comparison.

increased (i.e. become less negative). A homogeneous tie that closes a triangle is now more likely than one that does not close a triangle, while each is more likely than its heterogeneous counterparts.

8. Goodness of fit

Comparing a single outcome from the simulation to the original is of limited value. The logical next step, now that we have a model that may be starting to do a reasonable job of capturing our original network, is to simulate many networks and compare the full distribution of our statistic(s) of interest to that from the observed data. Let us continue with model 4, looking to see whether it does a better job of capturing the triangle count than model 2 did earlier. Let us use a large interval between our samples, in order to minimize the autocorrelation between consecutive samples:

```
R> sim4 <- simulate(model4, burnin = 1e+5, interval = 1e+5,
+    nsim = 100, verbose = TRUE, seed = 321)

Starting 100 MCMC iterations of 1e+05 steps each.
   TNT sampler accepted 32.637% of 100000 proposed steps.
   TNT sampler accepted 32.076% of 100000 proposed steps.
...
```

Notice that the number of steps accepted remains right at 33% or so throughout the chain of total length 10^7, another good sign that our model is not degenerate. Since we asked to simulate more than one network, `sim4` is of a different object class than `sim1` was:

```
R> class(sim4)

[1] "network.series"

R> names(sim4)

[1] "formula"  "networks" "stats"    "coef"

R> class(sim4$networks[[1]])
```

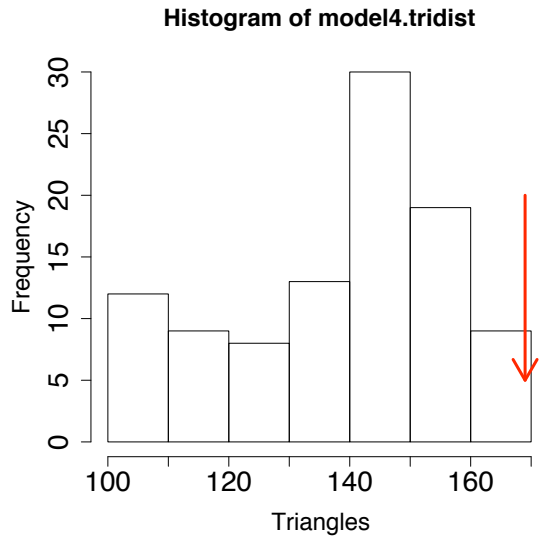

Figure 5: Number of triangles across 100 simulations from model 4 (histogram), compared to the original (arrow).

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

This output clarifies the R code that we will need to extract the number of triangles from each of the 100 networks:

```
R> model4.tridist <- sapply(sim4$networks, function(x) summary(x ~ triangle))
```

How does this distribution compare to the original?

```
R> hist(model4.tridist)
R> fmh.tri <- summary(fmh ~ triangle)
R> arrows(fmh.tri, 20, fmh.tri, 5, col = "red", lwd = 3)
```

You should see something like Figure 5. Next, you can check how many simulations were at least as extreme in the upper tail as our observed value, marked by the red arrow in Figure 5, using the command:

```
R> sum(fmh.tri <= model4.tridist)
```

```
[1] 1
```

In our simulation, the answer was 1 out of 100; our observed data evidently lie outside the central 95% of the simulated distribution, although model 4 is still a major improvement over model 2 with regard to triangles.

To make this process easier on the user, we have developed code to conduct all of the above stages in the goodness-of-fit automatically for some common network distributions; these include the degree distribution, the distribution of edgewise shared partners (the number of

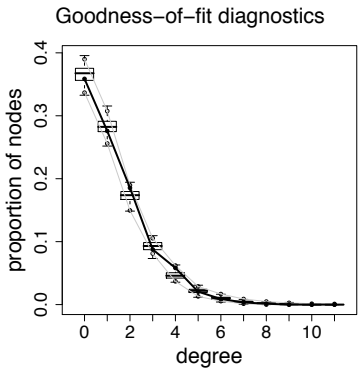

Figure 6: Model 4 goodness of fit for degree.

edges in which two friends have exactly k friends in common, for each value of k), and the geodesic distribution (the number of actor pairs for which the shortest path between them is of length k, for each value of k). Let us try the degree distribution.

```
R> gof4.deg <- gof(model4 ~ degree, verbose = TRUE, burnin = 1e+5,
+    interval = 1e+5, seed = 246)

Starting GOF for the given ERGM formula.
Calculating observed network statistics.
Starting simulations.
Sim 1 of 100 : Starting 1 MCMC iteration of 1e+05 steps.
TNT sampler accepted 32.673% of 100000 proposed steps.
...
```

(Some users may begin to encounter memory allocation problems at this point, depending on the system on which they are running. Cleaning up memory using the `rm()` command to remove unnecessary model fits, or the `gc()` command to force garbage collection, may help.)

This code simulates a number of networks (the default is 100), compiles the statistics for those 100 networks along with the original network, and calculates p-values. Although all of the information is stored in the `gof4.deg` object itself and can be explored directly, it is perhaps most easily examined (at least initially) through a simple plot of the data. To generate boxplots of the simulated statistics overlain with the original statistics, type:

```
R> plot(gof4.deg)
```

The resulting plot should resemble Figure 6. The dark black line represents the degree distribution for the original network. The boxplots represent the distribution across all 100 simulated networks of the proportion of vertices with the relevant degree, using R's defaults for boxplot construction. The soft lines connect the 95% bounds on these distributions. For the actual values, including associated p-values, simply type:

```
R> gof4.deg
```

```
Goodness-of-fit for degree

   obs min   mean max MC p-value
0  524 493 542.13 611       0.52
1  403 367 414.83 460       0.64
2  271 220 255.37 290       0.22
3  128 103 134.74 164       0.58
4   85  41  64.17  80       0.00
5   30  13  29.01  43       0.96
6   13   2  12.03  23       0.86
7    5   0   5.20  11       1.00
8    2   0   2.09   6       1.00
9    0   0   0.82   4       0.92
10   0   0   0.36   3       1.00
11   0   0   0.14   2       1.00
12   0   0   0.08   1       1.00
13   0   0   0.02   1       1.00
14   0   0   0.01   1       1.00
```

Note that without including information on degree directly, this model pretty well captures the degree distribution for this network.

Do be sure to use a sufficiently large interval on the `gof` command; otherwise one might obtain a plot that suggested that the model was well fitting, but in fact simply did not have enough time to move away from the original network. Here again we could see that the chain accepted about 33% of proposals, with 100,000 proposals between each run, meaning that consecutive samples differed by around 33,000 toggles.

Let us turn to edgewise shared partners and the geodesic distance distribution:

```
R> gof4.esp.dist <- gof(model4 ~ espartners + distance, verbose = TRUE,
+    burnin = 1e + 5, interval = 1e + 5, seed = 642)

Starting GOF for the given ERGM formula.
...
```

Note that since the simulated networks themselves are so large, they are not saved in the `gof` object by default; only those network statistics included in the formula for the `gof` call are. In practice it is often a good idea to include all three distributions (degree, edgewise shared partner, and geodesic distance) in one's formula even when one's primary interest is in a subset, if there is any chance one might be interested in the others later; this will avoid the need to regenerate the simulated networks.

We plot the edgewise shared partner and geodesic distributions on the log-odds scale (Figure 7), as this makes the latter in particular easier to visualize:

```
R> get(getOption("device"))(width = 8, height = 4)
R> par(mfrow = c(1, 2))
R> plot(gof4.esp.dist, plotlogodds = TRUE)
```

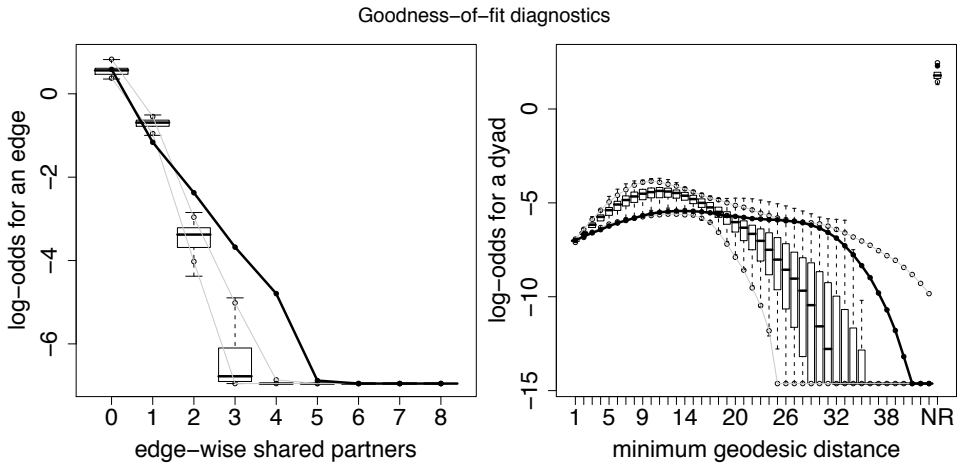

Figure 7: Model 4 goodness of fit for edgewise shared partner and geodesic distance.

Note that the first line of this code generates a graphics window of specified size regardless of operating system. The left-hand graph plots the edgewise shared partner distribution. The GWESP statistic was included in the model, which is a parametric form of the edgewise shared partner distribution, so one would hope that we are able to capture the distribution at least rudimentary. We seem to have done so, although our model overestimates the number of edges with one shared partner, and underestimates the number with two or more; it still does not capture the full amount of clustering.

On the other hand, we have included nothing about distance in our model, and distance is a very global property of the network. How well do the observed and simulated distributions match? What is the nature of the disparities? Can you think of ways to expand upon your model that might lead to a better global fit? We leave it to the reader to continue exploring different models to attempt to match the observed data as best as possible. For a full list of model terms available for this process, see Morris *et al.* (2008). Once you are done, see the Appendix to find out the actual model that was used to generate the network.

9. Next steps

If this tutorial has left you interested in learning about the remaining functionality of the `statnet` suite of packages, we encourage you to read the remaining papers in this issue, and the documentation available within each package. Also be aware that `statnet` is under continual development; users interested in keeping abreast of new features are encouraged to check the `statnet` homepage `http://statnetproject.org`, where they are also encouraged to sign up for the `statnet` mailing list.

Acknowledgments

The authors would like to acknowledge members of the **statnet** team, including Ryan Admiraal, Nicole Bohme, Susan Cassels, Krista Gile, Deven Hamilton, Aditya Khanna, Pavel Krivitsky, David Lockhart, James Moody, and Gail Potter. This work was funded by two

grants from the National Institutes of Health (R01-HD041877, R01-DA012831). DRH received additional funding from Le Studium, an agency of the Centre National de la Recherche Scientifique of France, and NIH grant R01-GM083603-01.

References

Butts CT (2008a). "**network**: A Package for Managing Relational Data in R." *Journal of Statistical Software*, **24**(2). URL http://www.jstatsoft.org/v24/i02/.

Butts CT (2008b). "Social Network Analysis with **sna**." *Journal of Statistical Software*, **24**(6). URL http://www.jstatsoft.org/v24/i06/.

Goodreau SM (2007). "Advances in Exponential Random Graph (p^*) Models Applied to a Large Social Network." *Social Networks*, **29**(2), 231–248.

Goodreau SM, Kitts J, Morris M (2008). "Birds of a Feather, or Friend of a Friend? Using Exponential Random Graph Models to Investigate Adolescent Social Networks." *Demography*, **45**. Forthcoming.

Handcock MS (2003a). "Assessing Degeneracy in Statistical Models of Social Networks." *Working Paper 39*, Center for Statistics and the Social Sciences, University of Washington. URL http://www.csss.washington.edu/Papers/.

Handcock MS (2003b). "Statistical Models for Social Networks: Inference and Degeneracy." In R Breiger, K Carley, P Pattison (eds.), "Dynamic Social Network Modeling and Analysis," volume 126, pp. 229–252. Committee on Human Factors, Board on Behavioral, Cognitive, and Sensory Sciences, National Academy Press, Washington, DC.

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2008). "**statnet**: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data." *Journal of Statistical Software*, **24**(1). URL http://www.jstatsoft.org/v24/i01/.

Harris KM, Florey F, Tabor J, Bearman PS, Jones J, Udry JR (2003). "The National Longitudinal Study of Adolescent Health: Research Design." *Technical report*, University of North Carolina. URL http://www.cpc.unc.edu/projects/addhealth/design/.

Hunter DR (2007). "Curved Exponential Family Models for Social Networks." *Social Networks*, **29**, 216–230.

Hunter DR, Goodreau SM, Handcock MS (2008a). "Goodness of Fit for Social Network Models." *Journal of the American Statistical Association*, **103**, 248–258.

Hunter DR, Handcock MS, Butts CT, Goodreau SM, Morris M (2008b). "**ergm**: A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks." *Journal of Statistical Software*, **24**(3). URL http://www.jstatsoft.org/v24/i03/.

Morris M, Handcock MS, Hunter DR (2008). "Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects." *Journal of Statistical Software*, **24**(4). URL http://www.jstatsoft.org/v24/i04/.

Plummer M, Best N, Cowles K, Vines K (2007). ***coda****: Output Analysis and Diagnostics for MCMC*. R package version 0.13-1, URL http://CRAN.R-project.org/package=coda.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, Version 2.6.1, URL http://www.R-project.org/.

Robins G, Morris M (2007). "Advances in Exponential Random Graph (p^*) Models." *Social Networks*, **29**(2), 169–172.

Robins G, Pattison P, Kalish Y, Lusher D (2007a). "An Introduction to Exponential Random Graph (p^*) Models for Social Networks." *Social Networks*, **29**(2), 173–191.

Robins G, Snijders T, Wang P, Handcock M, Pattison P (2007b). "Recent Developments in Exponential Random Graph (p^*) Models for Social Networks." *Social Networks*, **29**(2), 192–215.

Snijders TAB, Pattison P, Robins GL, Handcock MS (2006). "New Specifications for Exponential Random Graph Models." *Sociological Methodology*, **36**, 99–153.

Udry JR (2003). "The National Longitudinal Study of Adolescent Health (Add Health), Waves I & II, 1994–1996; Wave III, 2001–2002." *Technical report*, Carolina Population Center, University of North Carolina at Chapel Hill.

Wasserman SS, Pattison P (1996). "Logit Models and Logistic Regressions for Social Networks: I. An Introduction to Markov Graphs and p^*." *Psychometrika*, **61**(3), 401–425.

A. Creation of the Faux Magnolia High dataset

Faux Magnolia High is based upon schools 086 and 186 from the Add Health Wave I dataset. Add Health (officially the National Longitudinal Study of Adolescent Health) was a study of more than 90,000 US students in grades 7-12 obtained through a stratified sample of schools. The first wave was conducted in 1994-1995, and included a friendship nomination module based upon student rosters. Our school pair represents a junior and senior high school, and students were presented with a roster for both. For more information on Add Health, including instructions for obtaining public use copies of the network data, please see the Add Health Web page, http://www.cpc.unc.edu/addhealth/.

The process we used to develop Faux Magnolia High comprises the following steps:

1. Extract schools 086 and 186 into an R network object.
2. Remove all vertices that did not take the survey or were not on the school roster.
3. Convert the data from directed to undirected, by defining an undirected friendship X_{ij} as equaling 1 iff i named j as a friend and j named i as a friend.
4. Define race based on the answers to two questions (one on Hispanic ethnicity and one on race), allowing Hispanic identity as primary (i.e. all Hispanic are one category, non-Hispanic whites are another, non-Hispanic blacks are another, etc.)
5. Impute any missing Race, Grade and Sex values by random draw with weights determined by the size of the attribute classes in the school.
6. Fit the following model to the resulting school:

```
R> magnolia.fit <- ergm(magnolia ~ edges + absdiff("Grade") +
+    nodematch("Race", diff = TRUE) + nodematch("Grade", diff = TRUE) +
+    nodematch("Sex", diff = FALSE) + gwesp(0.25, fixed = TRUE),
+    burnin = 10000, interval = 1000, MCMCsamplesize = 2500, maxit = 25,
+    control = control.ergm(steplength = .25))
```

7. Simulate a single network from this model fit with fixed density:

```
R> faux.magnolia.high <- simulate(magnolia.fit, nsim = 1, burnin = 1e+8,
+    constraints = ~ edges)
```

Affiliation:

Steven M. Goodreau
Department of Anthropology
University of Washington
Campus Box 353100
Seattle, WA 98195, United States of America
E-mail: goodreau@u.washington.edu
URL: http://faculty.washington.edu/goodreau

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 24, Issue 9 *Submitted:* 2007-06-01
February 2008 *Accepted:* 2007-12-25