Journal of Statistical Software

February 2008, Volume 24, Issue 4. *http://www.jstatsoft.org/*

Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects

Martina Morris
University of Washington

Mark S. Handcock
University of Washington

David R. Hunter
Penn State University

Abstract

Exponential-family random graph models (ERGMs) represent the processes that govern the formation of links in networks through the terms selected by the user. The terms specify network statistics that are sufficient to represent the probability distribution over the space of networks of that size. Many classes of statistics can be used. In this article we describe the classes of statistics that are currently available in the **ergm** package. We also describe means for controlling the Markov chain Monte Carlo (MCMC) algorithm that the package uses for estimation. These controls affect either the proposal distribution on the sample space used by the underlying Metropolis-Hastings algorithm or the constraints on the sample space itself. Finally, we describe various other arguments to core functions of the **ergm** package.

Keywords: exponential-family random graph model, Markov chain Monte Carlo, maximum likelihood estimation, p-star model.

1. Introduction

The terms in an exponential-family random graph model (ERGM) are somewhat different than those in a traditional statistical model. In a traditional model, the data consist of a set of observations where each observation has been measured on a number of separate variables. One or more of these variables becomes the response variable (say, income), and the others are used as predictors (say, race and education). While these variables may be correlated, they have been measured separately for each unit. In a network, the observations also consist of a response variable — the state of a dyad, or pair of nodes, typically measured by the presence or absence of a tie, or ties in the case of a directed network, between them — and separately measured attributes of the nodes, the dyads and the ties (for example, the age of the nodes, whether the dyad is a sibling pair, and the duration of measured tie). But in an

ERGM for this network, the predictors are functions of the ties themselves. These predictors, called "network statistics," represent configurations of ties (for example, triangles, or triangles of three nodes of the same age) that are hypothesized to occur more often or less often than expected by chance. As these predictors are direct functions of the response variable, ERGMs can be thought of as autoregressive models (see, e.g., autologistic models in Besag 1974), and this changes many aspects of model specification and estimation.

1.1. Model specification for ERGMs

The general classes of ERG models are defined by the terms included as predictors. Examples, given in Hunter, Handcock, Butts, Goodreau, and Morris (2008b), include dyadic independent, dyadic dependent, and curved exponential-family (CEF) terms. Each predictor in an ERGM represents a specific configuration of links, such as edges or triangles, and the term is a function of the number of such configurations in the network. (Note that we use "edge" and "tie" synonymously in this article.) In turn, these terms, with their coefficients, define the probability of each edge and the probability of the entire network. A trivial example is the homogeneous Bernoulli (or Erdős-Rényi) model: the configuration is an edge, the term is the total number of edges, and the network may be viewed as a collection of independent and identically distributed Bernoulli random variables.

Every predictor entered into an ERGM must have an algorithm for calculating the associated network statistic, or, more precisely, an algorithm for calculating its associated *change statistic*: The difference in the value of the network statistic for two networks that differ from each other only in the presence or absence of a proposed edge (or edges). For the simple Erdős-Rényi model above, the change statistic for adding any edge is always 1. Naturally, most other terms require more substantial computation.

Section 2 below describes the many commonly-used network statistics that are included in the **statnet** suite of packages (Handcock, Hunter, Butts, Goodreau, and Morris 2003) for R (R Development Core Team 2007) as part of the **ergm** package, which provides the core functionality for statistical modeling in the **statnet** suite. The list of available terms can also be obtained in R by typing `help("ergm.terms")` after loading the **ergm** package (usually by typing `library("statnet")`). These terms are used in calls to the `ergm` function, to fit an ergm model; calls to `simulate`, to simulate networks from an `ergm` model fit; and calls to `summary`, to obtain measurements of network statistics on a dataset. See Hunter *et al.* (2008b) and Goodreau, Handcock, Hunter, Butts, and Morris (2008a) for examples of these functions. The terms described below are also available to the other packages in the **statnet** suite.

Note that the terms currently included in the **ergm** package are a small subset of the possible terms that could be used in an ERGM. As time goes on, we will continue to add new terms, integrating them into the **statnet** documentation and updating the **statnet** Web site accordingly at `http://statnetproject.org/`. In addition, the **statnet** suite of packages has been written to allow users to code and use their own terms, via the **ergmuserterms** package. Coding requires some knowledge of ANSI standard C (Kernighan and Ritchie 1988) and some knowledge of R. Full details about the method for coding user terms can be found at the **statnet** Web site, and we strongly encourage interested users to use the facility to create their own terms!

Finally, this article is about which terms are available and which terms can and cannot be used in certain situations; the larger question of how to go about choosing terms wisely is

beyond its scope. There are some things we can say — for instance, as usual in statistical modeling, a user must avoid introducing linear dependencies among the predictors by, say, using both `edges` and `density` in the same model — but the effective choice of terms for an ERGM depends on theory and context. For that reason, it is important not to simply take terms used in one field (say social networks) and use them to represent processes in another field (say food webs). Unless there is an independent theoretical justification, this is likely to lead to degeneracy and/or nonsensical results. Readers interested in learning more about choosing terms in the context of social network modeling may wish to consult the recent special issue of *Social Networks* (Robins and Morris 2007) and the many references contained therein.

2. Terms used in exponential-family random graph models

What follows is a list of model terms currently available in the **ergm** package and a brief description of each. Terms are specified by an R formula object, giving the network and network statistics, of the form

```
y ~ <term1> + <term2> + ...
```

where `y` is a `network` object (Butts 2008a) or a matrix that can be coerced to a `network` object; and `<term 1>`, `<term 2>`, etc. are terms chosen from the list below. For example, after typing `data("faux.mesa.high")` to access the `network` object called `faux.mesa.high`, one could type

```
R> myformula <- faux.mesa.high ~ edges + nodematch("Grade", diff = TRUE)
R> mymodel <- ergm(myformula)
R> mymodel$coef
```

```
            edges  nodematch.Grade.7  nodematch.Grade.8  nodematch.Grade.9
        -6.034045           2.847142           2.914487           2.438521
nodematch.Grade.10 nodematch.Grade.11 nodematch.Grade.12
         2.557946           3.310430           3.731460
```

For details about interpreting the output above, see similar models in other articles in this journal volume — e.g., `model3` in Hunter *et al.* (2008b) or `model2` in Goodreau *et al.* (2008a) — and read the descriptions of the `edges` and `nodematch` terms below.

We group the terms below into categories that represent classes of generative model processes. This gives a sense of the modeling capabilities currently available in **statnet** and the alternative specifications that can be used to represent a class of generative processes. An alphabetical list follows in tabular format in Appendix A, with check-boxes showing key restrictions and functionality.

2.1. Basic terms

These terms control the overall probability of a link in directed and undirected networks.

- `edges` — **Edges:** This term adds one network statistic equal to the number of edges in the network. For undirected networks, `edges` is equal to `kstar(1)`; for directed networks, `edges` is equal to both `ostar(1)` and `istar(1)`.

- `density` — **Density:** This term adds one network statistic equal to the density of the network. For undirected networks, `density` equals `kstar(1)` or `edges` divided by $n(n-1)/2$; for directed networks, `density` equals `edges` or `istar(1)` or `ostar(1)` divided by $n(n-1)$.

- `mutual(attrname, diff = FALSE, keep = NULL)` — **Mutuality:** This term adds one network statistic to the model, equaling the number of pairs of actors i and j for which $(i \to j)$ and $(j \to i)$ both exist. This term can only be used with directed networks. If the optional `attrname` argument is used, only mutual pairs that match on the named vertex attribute are counted. The optional modifiers `diff` and `keep` are used in the same way as for the `nodematch` term; refer to this term for details and an example.

- `asymmetric(attrname, diff = FALSE, keep = NULL)` — **Asymmetric dyads:** This term adds one network statistic to the model, equaling the number of pairs of actors for which exactly one of $(i \to j)$ or $(j \to i)$ exists. This term can only be used with directed networks. If the optional `attrname` argument is used, only asymmetric pairs that match on the named vertex attribute are counted. The optional modifiers `diff` and `keep` are used in the same way as for the `nodematch` term; refer to this term for details and an example.

2.2. Nodal attribute effects

Two types of attribute effects are represented by terms below: main effects and interactions. Main effects can be specified in terms of either continuous (quantitative) covariates or discrete (categorical) factors. Interaction effects control patterns of mixing for categorical nodal attributes.

General terms for main effects

- `nodecov(attrname)` — **Main effect of a covariate:** The `attrname` argument is a character string giving the name of a quantitative (not categorical) attribute in the network's vertex attribute list. This term adds a single network statistic to the model equaling the sum of `attrname(i)` and `attrname(j)` for all edges (i, j) in the network. For categorical attributes, see `nodefactor`. Note that for directed networks, `nodecov` equals `nodeicov` plus `nodeocov`.

  Similar terms for directed networks: `nodeicov`, `nodeocov`.

- `nodefactor(attrname, base = 1)` — **Main effect of a factor attribute:** The `attrname` argument is a character string giving the name of a categorical attribute in the network's vertex attribute list. This term adds multiple network statistics to the model, one for each of (a subset of) the unique values of the `attrname` attribute. Each of these statistics gives the number of times a vertex with that attribute appears in an edge in the network. In particular, for edges whose endpoints both have the same

attribute value, this value is counted twice. To include all attribute values is usually not a good idea, because the sum of all such statistics equals twice the number of edges and hence a linear dependency would arise in any model also including `edges`. Thus, the `base` argument tells which value(s) (numbered in order according to the `sort` function) should be omitted. The default value, one, means that the smallest (i.e., first in sorted order) attribute value is omitted. For example, if the "fruit" factor has levels "orange", "apple", "banana", and "pear", then to add just two terms, one for "apple" and one for "pear", set "banana" and "orange" to the base (remember to sort the values first) by using `nodefactor("fruit", base = 2:3)`. For an analogous term for quantitative vertex attributes, see `nodecov`.

Similar terms for directed networks: `nodeifactor`, `nodeofactor`.
Similar terms for bipartite networks: `b1factor`, `b2factor`.

Interaction terms for nodal attribute based mixing

The most general term here is `nodemix`. It can represent any form of categorical attribute-based mixing. When mixing displays the classic assortative/disassortative or distance-based biases, more parsimonious terms are a better choice.

- `nodemix(attrname, base = NULL)` — **Nodal Attribute Mixing:** The `attrname` argument is a character string giving the name of a categorical attribute in the network's vertex attribute list. This term adds one network statistic to the model for each possible pairing of attribute values. The statistic equals the number of edges in the network in which the nodes have that pairing of values. In other words, this term produces one statistic for every entry in the mixing matrix for the attribute. The ordering of the attribute values is alphabetical (for nominal categories) or numerical (for ordered categories). The optional `base` argument is a vector of integers corresponding to the pairings that should not be included. If `base` contains only negative integers, then these integers correspond to the only pairings that should be included. By default (i.e., with `base = NULL` or `base = 0`), all pairings are included.

- `nodematch(attrname, diff = FALSE, keep = NULL)` — **Uniform homophily and differential homophily:** The `attrname` argument is a character string giving the name of an attribute in the network's vertex attribute list. When `diff = FALSE`, this term adds one network statistic to the model, which counts the number of edges (i, j) for which `attrname(i) = = attrname(j)`. When `diff = TRUE`, p network statistics are added to the model, where p is the number of unique values of the `attrname` attribute. The kth such statistic counts the number of edges (i, j) for which `attrname(i) = = attrname(j) = = value (k)`, where `value(k)` is the kth smallest unique value of the attribute. If set to non-NULL, the optional `keep` argument should be a vector of integers giving the values of `k` that should be considered for matches; other values are ignored (this works for both `diff = FALSE` and `diff = TRUE`. For instance, to add two statistics, counting the matches for just the 2nd and 4th categories, use `nodematch` with `diff = TRUE` and `keep = c(2,4)`.

- `absdiff(attrname)` — **Absolute difference:** The `attrname` argument is a character string giving the name of a quantitative attribute in the network's vertex attribute list.

Using this term results in adding one network statistic to the model equal to the sum of `abs(attrname[i] - attrname[j])` for all edges (i,j) in the network.

- `absdiffcat(attrname, base = NULL)`— **Categorical absolute difference:** The argument `attrname` is a character string giving the name of a quantitative attribute in the network's vertex attribute list. This term adds one statistic for every possible nonzero distinct value of `abs(attrname[i] - attrname[j])` in the network; the value of each such statistic is the number of edges in the network with the corresponding absolute difference. The optional `base` argument is a vector indicating which differences should be omitted from the model (i.e., treated like the zero-difference category). The `base` argument, if used, should contain indices, not differences themselves. For instance, if the possible values of `abs(attrname[i] - attrname[j])` are 0, 0.5, 3, 3.5, and 10, then to omit 0.5 and 10 one should set `base = c(1, 4)`. Note that this term should generally be used only when the quantitative attribute has a limited number of possible values; an example is the `"Grade"` attribute of the `faux.mesa.high` and `faux.magnolia.high` datasets.

- `smalldiff(attrname, cutoff)` — **Small difference:** The `attrname` argument is a character string giving the name of a quantitative attribute in the network's vertex attribute list and `cutoff` is any real number. This term adds one network statistic to the model, equal to the number of edges (i, j) for which `abs(attrname(i) - attrname(j))` is less than or equal to `cutoff`.

2.3. Relational attribute effects

Like nodes, dyads and ties can have attributes that affect the probability of the tie. Examples of dyadic attributes include type (e.g., kin vs. non-kin) and the existence of other ties between the nodes (referred to as "multiplexity" in the social network field). Examples of edge attributes include both dyadic attributes and edge-specific properties like duration.

- `dyadcov(x, attrname)` — **Dyadic covariate:** If the network is directed, `x` is either a (symmetric) matrix of covariates, one for each possible dyad (i, j), or an undirected network; if the latter, optional argument `attrname` provides the name of the quantitative edge attribute to use for covariate values (in this case, missing edges in `x` are assigned a covariate value of zero). This term adds three statistics to the model, each equal to the sum of the covariate values for all dyads occupying one of the three possible non-empty dyad states (mutual, upper-triangular asymmetric, and lower-triangular asymmetric dyads, respectively), with the empty or null state serving as a reference category. If the network is undirected, `x` is either a matrix of edgewise covariates, or a network; if the latter, optional argument `attrname` provides the name of the edge attribute to use for edge values. This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The `edgecov` and `dyadcov` terms are equivalent for undirected networks.

- `edgecov(x, attrname = NULL)` — **Edge covariate:** The `x` argument is either a square matrix of covariates, one for each possible edge in the network, covariates, or a network; if the latter, optional argument `attrname` provides the name of the quantitative edge

attribute to use for covariate values (in this case, missing edges in `x` are assigned a covariate value of zero). This term adds one statistic to the model, equal to the sum of the covariate values for each edge appearing in the network. The `edgecov` term applies to both directed and undirected networks. For undirected networks the covariates are also assumed to be undirected. The `edgecov` and `dyadcov` terms are equivalent for undirected networks.

2.4. Degree and star distributions

Degrees and stars are equivalent representations for the distribution of node-based edge counts, with different interpretations. Degree terms simply represent the frequency distribution for nodal degrees; each node counts only once. Star terms count the distribution of k-star configurations, with smaller stars embedded in larger ones. As a result, nodes may be counted more than once for different values of k. There are parametric linear combinations of the degree and star statistics as well as fully non-parametric versions of each.

Non-parametric forms

- `degree(d, attrname)` — **Degree:** The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the ith such statistic equals the number of nodes in the network of degree `d[i]`, i.e. with exactly `d[i]` edges. The term `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the degree count is the number of nodes with the same value of the attribute as the ego node. This term can only be used with undirected networks.

  Similar terms for directed networks: `idegree`, `odegree`.
  Similar terms for bipartite networks: `b1degree`, `b2degree`.

- `kstar(k, attrname)` — **k-Stars:** The `k` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `k`. The ith such statistic counts the number of distinct `k[i]`-stars in the network, where a k-star is defined to be a node N and a set of k different nodes $\{O_1, \ldots, O_k\}$ such that the ties $\{N, O_i\}$ exist for $i = 1, \ldots, k$. The optional argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is over the number of k-stars where all nodes have the same value of the attribute. This term can only be used for undirected networks.

  Similar terms for directed networks: `istar`, `ostar`.
  Similar terms for bipartite networks: `b1star`, `b2star`.

Parametric forms

- `gwdegree(decay, fixed = FALSE)` — **Geometrically weighted degree distribution:** This term adds one network statistic to the model equal to the weighted degree distribution with weight parameter `decay`. This is the version given as equation (14) in

Hunter (2007). See the "Remark" in Section 3 of that paper to see why it is used rather than the version given in Snijders, Pattison, Robins, and Handcock (2006). The optional argument `fixed` indicates whether the scale parameter `lambda` is to be fit as a curved exponential-family model (see Hunter and Handcock 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a CEF model. This term can only be used with undirected networks.

Similar terms for directed networks: `gwidegree`, `gwodegree`.
Similar terms for bipartite networks: `gwb1degree`, `gwb2degree`.

- `altkstar(lambda, fixed = FALSE)` — **alternating k-stars:** This term adds one network statistic to the model equal to a weighted alternating sequence of k-star statistics with weight parameter `lambda`. This is the version given in Snijders *et al.* (2006). The `gwdegree` and `altkstar` terms produce mathematically equivalent models, as long as they are used together with the `edges` [or `kstar(1)`] term. See Section 3 and especially equation (13) of Hunter (2007) for details. The optional argument `fixed` indicates whether the scale parameter `lambda` is to be fit as a curved exponential-family model (see Hunter and Handcock 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a CEF model. This term can only be used with undirected networks.

Special cases

- `meandeg` — **Mean vertex degree:** This term adds one network statistic to the model equal to the average degree of the vertices. Note that this term is a constant multiple of both `edges` and `density`.

- `isolates` — **Isolates:** This term adds one statistic to the model equal to the number of isolates in the network. For an undirected network, an isolate is defined to be any node with degree zero. For a directed network, an isolate is any node with both in-degree and out-degree equal to zero.

- `twopath` — **2-Paths:** This term adds one statistic to the model, equal to the number of 2-paths in the network. For a directed network this is defined as a pair of edges $(i \to j), (j \to k)$, where i and k must be distinct. That is, it is a directed path of length 2 from i to k via j. For directed networks a 2-path is also a mixed 2-star but the interpretation is usually different; see `m2star`. For undirected networks this is defined as a pair of edges $\{i, j\}$, $\{j, k\}$. That is, it is an undirected path of length 2 from i to k via j, also known as a 2-star.

- `m2star` — **Mixed 2-stars, a.k.a 2-paths:** This term adds one statistic to the model, equal to the number of mixed 2-stars in the network, where a mixed 2-star is a pair of distinct edges $(j \leftarrow i), (j \to k)$. A mixed 2-star is sometimes called a 2-path because it is a directed path of length 2 from i to k via j. However, in the case of a 2-path the focus is usually on the endpoints i and k, whereas for a mixed 2-star the focus is usually on the midpoint j. This term can only be used with directed networks; for undirected networks see `kstar(2)`. See also `twopath`.

- `concurrent(attrname)` — **Concurrent node count:** This term adds one network statistic to the model, equal to the number of nodes in the network with degree 2 or higher. The optional term `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is the number of nodes with ties to at least 2 other nodes with the same value for that attribute as the index node. This term can only be used with undirected networks.

  Similar terms for bipartite networks: `b1concurrent`, `b2concurrent`.

2.5. Triangles and higher order cycles

The dyadic dependence implied by cycles can easily lead to degenerate models. For triad-based clustering, models with simple `triangle` or `tripercent` terms are almost always degenerate. The shared partner terms (in Section 2.6) typically give much better results.

General terms

- `triangle(attrname)` — **Triangles:** This term adds one statistic to the model equal to the number of triangles in the network. For an undirected network, a triangle is defined to be any set $\{(i,j),(j,k),(k,i)\}$ of three edges. For a directed network, a triangle is defined as any set of three edges $(i \rightarrow j)$ and $(j \rightarrow k)$ and either $(k \rightarrow i)$ or $(k \leftarrow i)$. The former case is called a "transitive triple" and the latter is called a "cyclic triple", so in the case of a directed network, `triangle` equals `ttriple` plus `ctriple` — thus at most two of these three terms can be in a model. The optional argument `attrname` restricts the count to those triples of nodes with equal values of the vertex attribute specified by `attrname`.

- `tripercent(attrname)` — **Triangle percentage:** This term adds one statistic to the model equal to 100 times the ratio of the number of triangles in the network to the sum of the number of triangles and the number of 2-stars not in triangles (the latter is considered a potential but incomplete triangle). For the definition of triangle, see `triangle`. The optional argument `attrname` restricts the counts (both numerator and denominator) to those triples of nodes with equal values of the vertex attribute specified by `attrname`. This term can only be used with undirected networks; for directed networks, it is difficult to define the numerator and denominator in a consistent and meaningful way.

- `cycle(k)` — **Cycles:** The `k` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `k`; the ith such statistic equals the number of cycles in the network with length exactly `k[i]`. The cycle statistic applies to both directed and undirected networks. For directed networks, it counts directed cycles of length k, as opposed to undirected cycles in the undirected case. The directed cycle terms of lengths 2 and 3 are equivalent to `mutual` and `ctriple` (respectively). The undirected cycle term of length 3 is equivalent to `triangle`, and there is no undirected cycle term of length 2.

Special cases

For a set of three nodes, there are two traditional ways to count edge configurations. The distinction between them is analogous to the difference between degrees and stars: one is a node-based count ("triads"), the other is edge-based ("triples"). The difference is easiest to see in the case of directed networks.

For triads, each set of three nodes counts only once, and it is classified by the set of edges, directed or undirected, that is present (analogous to degree counting). This set has 16 possible configurations in an unlabeled directed network. See Davis and Leinhardt (1972) for the list of possible configurations, or type `help("triad.classify")` after loading the **sna** package (Butts 2008b) to see the list in R. The frequency distribution of these configurations is called a "triad census."

For triples, each distinct set of three edges forming a triangle, regardless of the orientations of the three edges, is counted and classified (analogous to star counting). In an unlabeled directed network there are two possible configurations for a triple — a transitive triple ($i \to j, j \to k, i \to k$) and a cyclic triple ($i \to j, j \to k, k \to i$). Since the count is based on edge sets rather than node sets, each 3- node set can contribute up to eight triples: 0, 1, 2, 3, or 6 transitive triples, plus 0, 1, or 2 cyclic triples (see `ctriple` and `ttriple` below).

In an undirected graph these two counts simplify: a triad census counts the number of triads having 0, 1, 2, and 3 edges, and each set of three nodes contributes either zero or one triple (also called a triangle in this case).

The ergm package contains various terms counting triads, triples, and triangles, explained below.

- `ctriple(attrname)` — **Cyclic triples:** This term adds one statistic to the model, equal to the number of cyclic triples in the network, defined as a set of edges of the form $\{(i \to j), (j \to k), (k \to i)\}$. Note that for all directed networks, `triangle` is equal to `ttriple+ctriple`, so at most two of these three terms can be in a model. The argument `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is over the number of cyclic triples where all three nodes have the same value of the attribute. This term can only be used with directed networks.

- `ttriple(attrname)` — **Transitive triples:** This term adds one statistic to the model, equal to the number of transitive triples in the network, defined as a set of edges $\{(i \to j), (j \to k), (i \to k)\}$. Note that `triangle` equals `ttriple+ctriple` for a directed network, so at most two of the three terms can be in a model. The optional `attrname` is a character string giving the name of an attribute in the network's vertex attribute list. If this is specified then the count is over the number of transitive triples where all three nodes have the same value of the attribute. This term can only be used with directed networks.

- `intransitive` — **Intransitive triads:** This term adds one network statistic to the model equaling the number of triads in the network that are intransitive. The intransitive triads are those of type `111D`, `201`, `111U`, `021C`, or `030C` in the categorization of Davis and Leinhardt (1972). For details on the 16 possible triad types, see `?"triad.classify"` in the `sna` package. Note the distinction from the `ctriple` term. This term can only be used with directed networks.

- `transitive` — **Transitive triads:** This term adds one statistic to the model, equal to the number of transitive triads in the network. These are defined as the triads of type `120D`, `030T`, `120U`, or `300` in the categorization of Davis and Leinhardt (1972). For details on the 16 possible triad types, see `?"triad.classify"` in the `sna` package. Note the distinction from the `ttriple` term. This term can only be used with directed networks.

- `triadcensus(d)` — **Triad census:** For an undirected network, this term adds one network statistic for each of an arbitrary subset of the 16 possible types of triads categorized by Davis and Leinhardt (1972) as `003, 012, 102, 021D, 021U, 021C, 111D, 111U, 030T, 030C, 201, 120D, 120U, 120C, 210,` and `300`. Note that at least one category should be dropped; otherwise a linear dependency will exist among the 16 statistics, since they must sum to the total number of three-node sets. By default, the category `003`, which is the category of completely empty three-node sets, is dropped. This is considered category zero, and the others are numbered 1 through 15 in the order given above. By specifying a numeric vector of integers from 0 to 15 as the `d` argument, the user may specify a set of terms to add other than the default value of `1:15`. Each statistic is the count of the corresponding triad type in the network. For details on the 16 types, see `?"triad.classify"` in the `sna` package, on which this code is based. For an undirected network, the triad census is over the four types defined by the number of ties (i.e., 0, 1, 2, and 3), and the default is to add `1:3`, which is to say that the 0 is dropped; however, this too may be controlled by changing the `d` argument to a numeric vector giving a subset of $\{0, 1, 2, 3\}$.

- `balance` — **Balanced triads:** This term adds one network statistic to the model equaling the number of triads in the network that are balanced. The balanced triads are those of type `102` or `300` in the categorization of Davis and Leinhardt (1972). For details on the 16 possible triad types, see `?"triad.classify"` in the `sna` package. For an undirected network, the balanced triads are those with an even number of ties (i.e., 0 and 2).

- `localtriangle(attrname)` — **Triangles within neighborhoods:** This term adds one statistic to the model equal to the number of triangles in the network between nodes close to each other. For an undirected network, a local triangle is defined to be any set of three edges between nodal pairs $\{(i, j), (j, k), (k, i)\}$ that are in the same neighborhood. For a directed network, a triangle is defined as any set of three edges $(i \rightarrow j)$, $(j \rightarrow k)$ and either $(k \rightarrow i)$ or $(k \leftarrow i)$ where again all nodes are within the same neighborhood. The argument `attrname` is a network or an adjacency matrix that specifies whether the two nodes are in the same neighborhood. Note that `triangle`, with or without an argument, is a special case of `localtriangle`.

- `simmelian` — **Simmelian triads:** This term adds one statistic to the model equal to the number of Simmelian triads, as defined by Krackhardt and Handcock (2007). This is a complete sub-graph of size three. This term can only be used with directed networks.

- `simmelianties` — **Ties in Simmelian triads:** This term adds one statistic to the model equal to the number of ties in the network that are associated with Simmelian triads, as defined by Krackhardt and Handcock (2007). Each Simmelian has six ties

in it but, because Simmelians can overlap in terms of nodes (and associated ties), the total number of ties in these Simmelians is less than six times the number of Simmelians. Hence this is a measure of the clustering of Simmelians (given the number of Simmelians). This term can only be used with directed networks.

- `nearsimmelian` — **Near Simmelian triads:** This term adds one statistic to the model equal to the number of near Simmelian triads, as defined by Krackhardt and Handcock (2007). This is a sub-graph of size three which is exactly one tie short of being complete. This term can only be used with directed networks.

2.6. Shared partner distributions

New specifications for non-degenerate models of triad-based clustering have been developed (Snijders *et al.* 2006; Hunter and Handcock 2006) and used with some success (Hunter, Goodreau, and Handcock 2008a; Goodreau, Kitts, and Morris 2008b). These are dyad-based configurations, rather than node-based, and count the number of times that both nodes have a tie to a third node. One version restricts the count to base dyads with an edge, the other does not. Both are implemented in **ergm**. As with the degree and star terms, these statistics exist both in the form of parametric linear combinations and in non-parametric forms.

Non-parametric forms

- `dsp(d)` — **Dyadwise shared partners:** The `d` argument is a vector of distinct integers. This term adds one network statistic to the model for each element in `d`; the ith such statistic equals the number of dyads in the network with exactly `d[i]` shared partners. This term can be used with directed and undirected networks. For directed networks, the geometric weighting is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the dyad).

- `esp(d)` — **Edgewise shared partners:** This is just like the `dsp` term, except this term adds one network statistic to the model for each element in `d` where the ith such statistic equals the number of *edges* (rather than dyads) in the network with exactly `d[i]` shared partners. This term can be used with directed and undirected networks. For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the edge and in the same direction).

Parametric forms

The `gwdsp` and `gwesp` terms use a curved exponential family form to represent the shared partner distributions. See Hunter and Handcock (2006) for a general explanation of curved exponential-family models for networks and Hunter (2007) for more about these specific terms.

- `gwdsp(alpha, fixed = FALSE)` — **Geometrically weighted dyadwise shared partner distribution:** This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with weight parameter `alpha`. The optional argument `fixed` indicates whether the scale parameter `lambda` is

to be fit as a curved exponential-family model (see Hunter and Handcock 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a CEF model. This term can be used with directed and undirected networks. For directed networks the count is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the dyad).

- `gwesp(alpha, fixed = FALSE)` — **Geometrically weighted edgewise shared partner distribution:** This term is just like `gwdsp` except it adds a statistic equal to the geometrically weighted *edgewise* (not dyadwise) shared partner distribution with weight parameter `alpha`. The optional argument `fixed` indicates whether the scale parameter `lambda` is to be fit as a curved exponential-family model (see Hunter and Handcock 2006). The default is `FALSE`, which means the scale parameter is not fixed and thus the model is a CEF model. This term can be used with directed and undirected networks. For directed networks the geometric weighting is over homogeneous shared partners only (i.e., only partners on a directed two-path connecting the nodes in the edge and in the same direction).

2.7. Actor-specific effects

Each of the following terms adds a single statistic to the model for each node in the network.

- `receiver(base = 1)` — **Receiver effect:** This term adds one network statistic for each node equal to the number of in-ties for that node. This measures the popularity of the node. The term for the first node is omitted by default because of linear dependence that arises if this term is used together with `edges`, but its coefficient can be computed as the negative of the sum of the coefficients of all the other actors. That is, the average coefficient is zero, following the Holland-Leinhardt parametrization of the p_1 model (Holland and Leinhardt 1981). The `base` argument allows the user to determine which nodes' statistics should be omitted. The `base` argument can also be a vector of negative indices, to specify which should be added instead of deleted, and `base = 0` specifies that all statistics should be included. This term can only be used with directed networks. For undirected networks, see `sociality`.

- `sender(base = 1)` — **Sender effect:** This term is just like `receiver` but for out-ties instead of in-ties.

- `sociality(attrname, base = 1)` — **Undirected degree:** This term adds one network statistic for each node equal to the number of ties of that node. The optional `attrname` argument is a character string giving the name of an attribute in the network's vertex attribute list that takes categorical values. If provided, this term only counts ties between nodes with the same value of the attribute (an actor-specific version of the `nodematch` term). This term can only be used with undirected networks. For directed networks, see `sender` and `receiver`. By default, `base = 1` means that the statistic for the first node will be omitted, but this argument may be changed to control which statistics are included just as for the `sender` and `receiver` terms.

2.8. Whole network operators

In general, the philosophy of **statnet** modeling is generative: micro-level processes are proposed (via the model) to reproduce the macro-level network structure, and tested by goodness of fit. For simulation purposes, however, additional control over the network structure may be desired. The following special purpose terms control the distance of a network from a target network

- `hamming(x, cov, attrname)` — **Hamming distance:** This term adds one statistic to the model equal to the weighted or unweighted Hamming distance of the network from the network specified by `x`. (If no argument is given, `x` is taken to be the observed network, i.e., the network on the left side of the $\sim$ in the formula that defines the ERGM.) Unweighted Hamming distance is defined as the total number of pairs (i, j) (ordered or unordered, depending on whether the network is directed or undirected) on which the two networks differ. If the optional argument `cov` is specified, then the weighted Hamming distance is computed instead, where each pair (i, j) contributes a pre-specified weight toward the distance when the two networks differ on that pair. The argument `cov` is either a matrix of edgewise weights or a network; if the latter, the optional argument `attrname` provides the name of the edge attribute to use for weight values.
- `hammingmix(attrname, x, base = 0, contrast = FALSE)` — **Hamming distance for a mixing matrix:** This term adds one statistic to the model for every possible pairing of attribute values of the network. Each such statistic is the Hamming distance (i.e., the number of differences) between the appropriate subset of dyads in the network and the corresponding subset in `x`. The ordering of the attribute values is alphabetical. If the option `contrast = TRUE` is used, then a statistic for the first pairing is not included, making it the de facto reference category. The option `base` gives the index of statistics to be omitted from the tabulation. For example `base = 2` will omit the second statistic, making it the de facto reference category.

3. Constraining the set of possible networks

The complete specification of a stochastic model for a network must include the support, which is the space of all possible realizable networks. Often this space is implicitly defined, given a particular set of nodes and a network of a particular type (e.g., directed or undirected), as the set of all possible networks on these nodes of this type. Another implicit constraint is the fact that some networks are bipartite (see the `is.bipartite` function in the **network** package), which means that all edges are *between* two predefined groups of nodes, never *within* these groups.

In its network-simulation routines, the **ergm** package currently automatically employs these implicit constraints when applicable. There are even certain model terms that are restricted to certain implicit constraint conditions. Such restrictions are noted in the table in Appendix A. Note in the case of bipartite network terms that we use the neutral "b1" and "b2" to refer to the two categories of nodes of a bipartite network, since the more commonly used "actor" and "event" designations are not always appropriate (for instance, in a bipartite network representing heterosexual relationships between individuals).

In addition to these implicit constraints, it is common to enforce additional explicit constraints for scientific reasons. A typical example is restricting the sample space to the subset of all networks having a fixed number of edges. This is a useful constraint in the context of diffusion models, for example, where the scientific question is often whether a certain structural feature of the network (e.g., homophily or the degree distribution) influences diffusion conditional on mean degree. To specify the constant edges constraint in **ergm**, use the `constraints = ~edges` argument in calls to any of the functions that rely on Markov chain Monte Carlo (MCMC), namely, `ergm`, to fit an ERGM; `simulate`, to simulate networks from a fitted ERGM; and `gof`, to assess goodness of fit for an ERGM (all three functions are described in detail in Hunter *et al.* 2008b). For example, to simulate a random network from the ERGM called `mymodel` that we considered in Section 2, constrained to the space of all 203-edge undirected networks (like `faux.mesa.high`) on the same nodes as `faux.mesa.high`, we can type

```
R> mynetwork1 <- simulate(mymodel, constraints = ~ edges)

Warning: The model contains the edges term and the proposal constraints
hold edges constant.  This term will be ignored.
```

Note in this example that the `simulate` function recognizes that the `edges` term is irrelevant if we are holding the number of edges fixed, and it prints a warning message to this effect.

Here are the types of constraints that may be used in conjunction with MCMC routines:

- `.` or `NULL` A placeholder for no constraints: all networks of a particular size and type have non-zero probability. Cannot be combined with other constraints.
- `bd(attribs, maxout, maxin, minout, minin)` Constrain maximum and minimum vertex degree. See the "Placing Bounds on Degrees" section in the online help for `ergm` (by typing `?"ergm"`) for more information. Note: For undirected networks, use only `maxout` and `minout`; both `maxin` and `minin` are ignored in this case.
- `degrees` and `nodedegrees` Preserve the degree of each vertex of the given network: only networks whose individual vertex degrees are the same as those in the network passed in the model formula have non-zero probability. This should only be used with undirected networks.
- `degreedist` Preserve the degree distribution of the network given: only networks whose overall degree distributions are the same as those in the network passed in the model formula have non-zero probability.
- `indegreedist` and `outdegreedist` Preserve the indegree or outdegree distribution, respectively, of the network given. These can only be used for directed networks.
- `edges` Preserve the edge count of the network given: only networks having the same number of edges as the network passed in the model formula have non-zero probability. This constraint works by proposes networks with a changed pair of dyads, one with an edge present and the other without. This ensures that all networks reached by the chain will have the same number of edges.

The constraints above should be passed in the right-hand-side of a formula without any left-hand-side (as in `~edges` above). More than one constraint may appear in such a formula, but not all possible combinations of the above constraints are supported. As an example of a combination of two constraints, we might wish to simulate a random undirected network on the `faux.mesa.high` nodes according to the `mymodel` model but conditional on having the same number of edges (203) *and* ensuring that no node has more than five edges. Then we could type

```
R> mynetwork2 <- simulate(mymodel, constraints = ~ edges + bd(maxout = 5))
```

```
Warning: The model contains the edges term and the proposal constraints
hold edges constant.  This term will be ignored.
Warning:  Initial network does not satisfy degree constraints.
 Proceeding anyway, but final network may not satisfy constraints.
```

However, when we do this, we get a warning saying that because the initial network (i.e., `faux.mesa.high`, which is part of the formula called `mymodel$formula`) does not satisfy the constraints, there is no guarantee that the final network will either. Therefore, in order to really carry out the stated goal, we need to replace `faux.mesa.high` by another network having the same number of nodes and edges, along with the same nodal attributes, but satisfying the desired constraint. This is sometimes tricky, but in this case (since there are 205 nodes but only 203 edges), it is very easy to connect node i with only node $i + 2$ for $1 \leq i \leq 203$. The first two lines below copy all of the nodes and attributes of the original network, then clear all of the edges, so that the new edges can be added:

```
R> mynetwork3 <- faux.mesa.high
R> mynetwork3[ , ] <- 0
R> for(i in 1:203) mynetwork3[i, i+2] <- mynetwork3[i+2, i] <- 1
```

This new network, `mynetwork3`, only has nodes with degree 1 or 2, so now we can use it as the initial network, along with the coefficient estimates from `mymodel`, to simulate a random network with the desired constraints:

```
R> mynetwork4 <- simulate(mynetwork3 ~ nodematch("Grade", diff = TRUE) +
+    edges, theta0 = mymodel$coef, constraints = ~ edges + bd(maxout = 5),
+    seed = 12345)
```

```
Warning: The model contains the edges term and the proposal constraints
hold edges constant.  This term will be ignored.
```

We may now check that the constraints were actually satisfied by noting that there are 203 edges in `mynetwork4` and that no node has degree higher than five. Note that the simulated network is random, but because the `seed` argument is used above, the output below should be obtained exactly:

```
R> summary(mynetwork4 ~ edges + degree(0:12))
```

```
   edges  degree0  degree1  degree2  degree3  degree4  degree5
     203       30       60       43       37       26        9
 degree6  degree7  degree8  degree9 degree10 degree11 degree12
       0        0        0        0        0        0        0
```

4. Computation considerations in the MCMC algorithm

The core of the computations in the **ergm** package is an MCMC algorithm that simulates a random sequence, or Markov chain, of networks in such a way that the probability that the chain is in any particular state after a large number of steps in the sequence is approximately given by an ERGM with a specified parameter value. The MCMC algorithm proceeds by comparing the probability of a new randomly selected "proposed" network to the current one in the chain, then deciding whether to accept the proposed network as the next step in the chain or not (if not, the chain remains at the same network for more than one step, and in either case the propose-compare-decide procedure is repeated). The core mathematical ideas behind this procedure, known as a Metropolis-Hastings algorithm, are described in Section 5 of Hunter *et al.* (2008b), but here we discuss some ways in which the MCMC algorithm can be fine-tuned.

The **ergm** package provides the user with a range of proposal modifiers and tuning parameters that can be used as arguments in calls to certain functions — `ergm`, `simulate`, and `gof`, all described in Hunter *et al.* (2008b) — that rely on MCMC. These arguments may be used to control things like the method of selecting proposal networks in the MCMC algorithm, the number of steps in the Markov chain, and the rate at which the chain should be sampled.

4.1. Speeding up search in the MCMC algorithm

Given the computationally intensive requirement to recalculate the change statistics for every term in the model for each proposed step of the MCMC estimation, an efficient algorithm is clearly desirable. One way to modify the Metropolis-Hastings MCMC algorithm is by changing the way it selects networks for proposing a change from the current state of the Markov chain. The most basic way to select proposal networks is to choose a dyad uniformly at random, then propose changing the network by toggling that dyad (changing it from an edge to a non-edge, or vice-versa).

We may modify this basic MCMC algorithm by leveraging known properties of the network to search for networks that are more likely. For instance, realistic networks are often fairly sparse. When using the basic MCMC algorithm for sparse networks, a high proportion of the proposed toggles are proposing to add an edge rather than take one away. Such proposals are often rejected by a model whose mean density is low, and so the Markov chain tends to spend multiple steps in the same state fairly often. Thus, it often makes sense to use the "TNT" (tie-no tie) sampler, which, rather than selecting a possible dyad to toggle uniformly at random, chooses an empty dyad with probability 1/2 (instead of the proportion of empty dyads at the current step, which is close to 1 for a sparse network). This bias in favor of proposing existing edges to toggle often leads to more "mixing" of the chain (i.e., fewer instances of staying in the same state for multiple steps of the Markov chain) than the basic MCMC algorithm, even though both algorithms lead to the same theoretical limiting distribution. In other words,

the TNT sampler often speeds convergence of the Markov chain relative to the basic uniform random sampler.

The TNT sampler is implemented in **ergm** by selecting `prop.weights = "TNT"` within the `control.ergm`, `control.simulate`, or `control.gof` functions; see the example below. Since TNT often works better in practice than the basic uniform random sampler, which may be invoked by selecting `prop.weights = "random"`, TNT is the default for many constraint settings. Thus, one may also select TNT by typing `prop.weights = "default"` or simply not typing anything. The `prop.weights` control options can be used in conjunction with constraints on the space of networks, when applicable, although not all possible combinations of constraints and samplers are implemented. For instance, the following command overrides the default TNT sampler used for the `bd` (bounded degree) constraint:

```
R> mynetwork5 <- simulate(mynetwork3 ~ nodematch("Grade", diff = TRUE) +
+    edges, theta0 = mymodel$coef, constraints = ~ bd(maxout = 5),
+    seed = 12345, control = control.simulate(prop.weights = "random"),
+    burnin = 1e + 5)
R> summary(mynetwork5 ~ edges + degree(0:12))

   edges  degree0  degree1  degree2  degree3  degree4  degree5
     178       30       59       72       28       11        5
 degree6  degree7  degree8  degree9 degree10 degree11 degree12
       0        0        0        0        0        0        0
```

Note that in the example above, we do not include the `edges` constraint because this constraint (with or without the `db` constraint) only allows one possible value of `prop.weights` in the current version of **ergm**, i.e., the default, which is a version of `"random"`.

4.2. Markov chain length and sampling rate

Aside from controlling how the proposal networks are chosen at each step of the Markov chain, it is possible to control certain other characteristics of the chain. For instance, the total number of steps in the chain is determined by how many networks will ultimately be sampled from the chain and how many steps the chain is allowed to run between sampled networks. The number of networks to be sampled is specified by the user via the `nsim` argument for the `simulate` and `gof` functions, or the `MCMCsamplesize` argument for the `ergm` function.

The `burnin` and `interval` arguments, available for all three functions, control how many networks to ignore at the beginning of the chain and between sampled networks, respectively. The reason for the `interval` argument is to attempt to reduce the correlation between sets of network statistics from one sampled network to the next. Note that it is generally inadvisable to discard Markov chain iterations like this, since even correlated statistics contain information that is useful for estimation purposes. However, in this case there is a trade-off between the cost (in time) of more iterations of the Markov chain and the cost (in memory) of storing a large number of sampled networks. Because the iterations are generally very fast in **ergm** and correlation between neighboring states of the Markov chain is generally very high, it is worthwhile to discard large numbers of networks in order to reduce the correlation and therefore save storage space.

5. Other control options

Other arguments to the `ergm`, `simulate`, and `gof` functions do not affect the MCMC algorithm directly, but control other aspects of the calculations instead. We only describe a few of these arguments here. For complete lists of available arguments, check the R help files for these functions. Furthermore, to learn a bit more about how they are actually used, consult Hunter *et al.* (2008b) and Goodreau *et al.* (2008a) in this volume.

5.1. Maximum likelihood estimation fine tuning

The `ergm` function, which uses a stochastic algorithm for maximum likelihood estimation of the ERGM parameters, allows a certain degree of control over the estimation procedure itself. Some of these capabilities are still the subject of ongoing research, while others are more well-established. The `maxit` argument may be used to control the number of times that the stochastic algorithm is restarted with parameter values equal to the current "best guess" at the maximum likelihood estimator. There are also several possible methods in the literature for carrying out maximum likelihood estimation via a stochastic algorithm. By default, a Newton-Raphson method is used [`control = control.ergm(style = "Newton-Raphson")`], but it is also possible to select a form of stochastic approximation called Robbins-Monro [using `control = control.ergm(style = "Robbins-Monro")`]. See Snijders (2002) or Snijders and van Duijn (2002) for details on the Robbins-Monro algorithm for ERGM estimation.

5.2. Miscellaneous

All three functions — `ergm`, `simulate`, and `gof` — allow the user to specify a seed for the random number generator via the `seed` argument. This enables the exact duplication of stochastic results from trial to trial in case such duplication is helpful. Furthermore, the `ergm` function has a `force.mcmc` argument that forces the use of a stochastic MCMC estimation algorithm even for dyadic independence models, for which the maximum likelihood estimator may be obtained exactly using maximum pseudolikelihood estimation (MPLE). See Hunter *et al.* (2008b) for details about MPLE. The `summarizestats` argument, available to `gof` and to `simulate`, may be set to TRUE if one desires a summary of the sufficient statistics of the sampled network(s). This can useful as a diagnostic tool.

Note that certain arguments to these three functions, including some discussed above, must be passed to them using the `control.ergm`, `control.simulate` and `control.gof` functions, as follows:

```
R> ergm(..., control = control.ergm(...))
R> simulate(..., control = control.simulate(...))
R> gof(..., control = control.gof(...))
```

Typing `help("control.ergm")`, `help("control.simulate")`, or `help("control.gof")` will display the respective online help documents, which give lists of the available options.

Because **statnet** is constantly being improved, additional arguments will continue to be added. Interested users are encouraged to check the **statnet** project Web site, which is located at http://statnetproject.org/, and/or join the email list for users of **statnet**.

Acknowledgments

The authors would like to acknowledge members of the **statnet** team, including Ryan Admiraal, Nicole Bohme, Carter Butts, Susan Cassels, Krista Gile, Steven Goodreau, Deven Hamilton, Aditya Khanna, Pavel Krivitsky, David Lockhart, and James Moody. 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

Besag J (1974). "Spatial Interaction and the Statistical Analysis of Lattice Systems." *Journal of the Royal Statistical Society B*, **36**, 192–236.

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/.

Davis JA, Leinhardt S (1972). "The Structure of Positive Interpersonal Relations in Small Groups." In J Berger (ed.), "Sociological Theories in Progress, Volume 2," pp. 218–251. Houghton Mifflin, Boston.

Goodreau SM, Handcock MS, Hunter DR, Butts CT, Morris M (2008a). "A **statnet** Tutorial." *Journal of Statistical Software*, **24**(9). URL http://www.jstatsoft.org/v24/i09/.

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

Handcock MS, Hunter DR, Butts CT, Goodreau SM, Morris M (2003). ***statnet**: Software Tools for the Statistical Modeling of Network Data*. Statnet Project http://statnetproject.org/, Seattle, WA. R package version 2.0, URL http://CRAN.R-project.org/package=statnet.

Holland PW, Leinhardt S (1981). "An Exponential Family of Probability Distributions for Directed Graphs." *Journal of the American Statistical Association*, **76**(373), 33–65.

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 (2006). "Inference in Curved Exponential Family Models for Networks." *Journal of Computational and Graphical Statistics*, **15**(3), 565–583.

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/.

Kernighan BW, Ritchie DM (1988). *The C Programming Language*. Prentice Hall Press, Upper Saddle River, NJ.

Krackhardt D, Handcock MS (2007). "Heider versus Simmel: Emergent Features in Dynamic Structures." In EM Airoldi (ed.), "Workshop on Statistical Network Analysis, ICML 2006, Pittsburgh, USA, June 29, 2006," volume 4503 of *Lecture Notes in Computer Science*, pp. 14–27. Springer-Verlag.

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.

Snijders TAB (2002). "Markov Chain Monte Carlo Estimation of Exponential Random Graph Models." *Journal of Social Structure*, **3**(2).

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

Snijders TAB, van Duijn MAJ (2002). "Conditional Maximum Likelihood Estimation under Various Specifications of Exponential Random Graph Models." In J Hagberg (ed.), "Contributions to Social Network Analysis, Information Theory, and Other Topics in Statistics; A Festschrift in honour of Ove Frank," pp. 117–134. University of Stockholm, Department of Statistics, Stockholm.

A. Table of model terms

	Term name (arguments)	Restricted to: Undirected only	Directed only	Bipartite only	Type of term: Dyadic independence	Categorical nodal attribute	Quantitative nodal attribute	Triad-related
1	`absdiff(attrname)`				×		×	
2	`absdiffcat(attrname, base = NULL)`				×	×		
3	`altkstar(lambda, fixed = FALSE)`	×						
4	`asymmetric (attrname, diff = FALSE, keep = NULL)`		×		(a)	(b)		×
5	`b1concurrent(attrname)`	×		×		(b)		
6	`b1degree(d, attrname)`	×		×		(b)		
7	`b1factor(attrname, base = 1)`	×		×	×	×		
8	`b1star(k, attrname)`	×		×		(b)		
9	`b2concurrent(attrname)`	×		×		(b)		
10	`b2degree(d, attrname)`	×		×		(b)		
11	`b2factor(attrname, base = 1)`	×		×	×	×		
12	`b2star(k, attrname)`	×		×		(b)		
13	`balance`							×
14	`concurrent(attrname)`	×				(b)		
15	`ctriple(attrname)`		×			(b)		×
16	`cycle(k)`							
17	`degree(d, attrname)`	×				(b)		
18	`density`				×			
19	`dsp(d)`							
20	`dyadcov(x, attrname)`				×			
21	`edgecov(x, attrname = NULL)`				×			
22	`edges`				×			
23	`esp(d)`							
24	`gwb1degree(decay, fixed = FALSE)`	×		×				
25	`gwb2degree(decay, fixed = FALSE)`	×		×				
26	`gwdegree(decay, fixed = FALSE)`	×						
27	`gwdsp(alpha, fixed = FALSE)`							
28	`gwesp(alpha, fixed = FALSE)`							
29	`gwidegree(decay, fixed = FALSE)`		×					
30	`gwodegree(decay, fixed = FALSE)`		×					

Table 1: Summary of **ergm** model terms. Terms that are not restricted to a particular type of network with an × in columns 1 through 3 may apply to any network. (a): Term makes dyads independent, but the pairs (i, j) and (j, i) are dependent so MCMC is used. (b): Optional argument allows term to count only structures that match on a categorical attribute.

	Term name (arguments)	Restricted to: Undirected only	Directed only	Bipartite only	Type of term: Dyadic independence	Categorical nodal attribute	Quantitative nodal attribute	Triad-related
31	`hamming(x, cov, attrname)`				×			
32	`hammingmix(attrname, x, base = 0, contrast = FALSE)`				×	×		
33	`idegree(d, attrname)`		×			(b)		
34	`intransitive`		×					×
35	`isolates`							
36	`istar(k, attrname)`		×			(b)		
37	`kstar(k, attrname)`	×				(b)		
38	`localtriangle(x)`							×
49	`m2star`		×					
40	`meandeg`				×			
41	`mutual(attrname, diff = FALSE, keep = NULL)`		×		(a)	(b)		
42	`nearsimmelian`		×					×
43	`nodecov(attrname)`				×		×	
44	`nodefactor(attrname, base = 1)`				×	×		
45	`nodeicov(attrname)`		×				×	
46	`nodeifactor(attrname, base = 1)`		×		×	×		
47	`nodematch(attrname, diff = FALSE, keep = NULL)`				×	×		
48	`nodemix(attrname, base = NULL)`				×	×		
49	`nodeocov(attrname)`		×				×	
50	`nodeofactor(attrname, base = 1)`		×		×	×		
51	`odegree(d, attrname)`		×			(b)		
52	`ostar(k, attrname)`		×			(b)		
53	`receiver(base = 1)`		×		×			
54	`sender(base = 1)`		×		×			
55	`simmelian`		×					×
56	`simmelianties`		×					×
57	`smalldiff(attrname, cutoff)`				×		×	
58	`sociality(attrname, base = 1)`	×				(b)		
59	`transitive`		×					×
60	`triadcensus(d)`							×
61	`triangle(attrname)`					(b)		×
62	`tripercent(attrname)`	×				(b)		×
63	`ttriple(attrname)`		×			(b)		×
64	`twopath`							

Table 2: Summary of **ergm** model terms. Terms that are not restricted to a particular type of network with an × in columns 1 through 3 may apply to any network. (a): Term makes dyads independent, but the pairs (i, j) and (j, i) are dependent so MCMC is used. (b): Optional argument allows term to count only structures that match on a categorical attribute.

Affiliation:

Martina Morris
Departments of Sociology and Statistics
University of Washington
Seattle, WA 98195, United States of America
E-mail: morrism@u.washington.edu
URL: http://faculty.washington.edu/morrism/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 24, Issue 4 *Submitted:* 2007-06-01
February 2008 *Accepted:* 2007-12-25