Advanced Features of the ergm Package for Modeling Networks

Statnet Development Team

Contents

The Statnet Project
Introduction to this workshop/tutorial.
1. Brief review of ERGM modeling framework
2. Sample space constraints
3. Tuning ERGM estimation
4. Term operators
5. Modeling multiple networks
6. Estimation in the presence of missing data
7. Multilayer networks
References

Last updated 2024-06-23

This tutorial is a joint product of the Statnet Development Team:

Pavel N. Krivitsky (University of New South Wales)
Martina Morris (University of Washington)
Mark S. Handcock (University of California, Los Angeles)
Carter T. Butts (University of California, Irvine)
David R. Hunter (Penn State University)
Steven M. Goodreau (University of Washington)
Chad Klumb (University of Washington)
Skye Bender de-Moll (Oakland, CA)
Michał Bojanowski (Kozminski University, Poland)

The Statnet Project

All Statnet packages are open-source, written for the R computing environment, and published on CRAN. The source repositories are hosted on GitHub. Our website is statnet.org

- Need help? For general questions and comments, please email the Statnet users group at statnet_help@uw.edu. You’ll need to join the listserv if you’re not already a member. You can do that here: Statnet_help listserv.
- Found a bug in our software? Please let us know by filing an issue in the appropriate package GitHub repository, with a reproducible example.
- Want to request new functionality? We welcome suggestions – you can make a request by filing an issue on the appropriate package GitHub repository. The chances that this functionality will be developed are substantially improved if the requests are accompanied by some proposed code (we are happy to review pull requests).
- For all other issues, please email us at contact@statnet.org.
Introduction to this workshop/tutorial.

This workshop and tutorial cover advanced topics in modeling network data with *Exponential family Random Graph Models* (ERGMs) using *statnet* software. It assumes a familiarity with the *ergm* package at least at the level of the introductory workshop entitled ERGMs using statnet. This online tutorial is also designed for self-study, with example code and self-contained data.

Some of the material in this workshop is drawn from a pair of recent articles by Krivitsky et al.: *ergm* 4: New features and *ergm* 4: Computational Improvements.

Prerequisites

This workshop assumes basic familiarity with *R*, experience with network concepts and data, familiarity with the ERGM modeling framework, and proficiency using the *ergm* package.

Software installation

Minimally, you will need to install the latest version of *R* (available here) and the *statnet* packages *ergm.multi* and *ggrepel*, which will in turn pull in the rest of the packages needed to run the code presented here.

The workshops are conducted using the free version of *Rstudio* (available here).

If you have not already downloaded the *statnet* packages for this workshop, the quickest way to install these (and the other most commonly used packages from the *statnet* suite), is to open an *R* session and type:

```r
install.packages(c('ergm.multi', 'ggrepel'))
update.packages(ask=FALSE, checkBuilt=TRUE)
```

The first line above will install all three required packages, since both *ergm.multi* depends on both *ergm* and *network*. The second line ensures that any already-installed packages, including *ergm* and *network*, are updated to their latest versions.

Now load *ergm.multi*, which will also load *ergm* and others.

```r
library(ergm.multi)
```

Loading required package: *ergm*

Loading required package: *network*

*network* 1.18.2 (2023-12-04), part of the Statnet Project
*news(package="network")' for changes since last version
*citation("network")' for citation information
*https://statnet.org' for help, support, and other information

*ergm* 4.6.0 (2023-12-17), part of the Statnet Project
*news(package="ergm")' for changes since last version
*citation("ergm")' for citation information
*https://statnet.org' for help, support, and other information

*ergm* 4 is a major update that introduces some backwards-incompatible changes. Please type *news(package="ergm")'* for a list of major changes.

*ergm.multi* 0.2.1 (2024-02-20), part of the Statnet Project
Throughout, we will set a random seed via `set.seed()` for commands in tutorial that require simulating random values. This is not necessary, but it ensures that you will get the same results as the online tutorial.

1. Brief review of ERGM modeling framework

The general form of an ERGM can be written as

\[ P(Y = y) = \frac{\exp(\theta^\top g(y))}{k(\theta)}, \]

where

- \( Y \) is the random variable for the state of the network (with realization \( y \)),
- \( g(y) \) is a \( p \)-dimensional vector of model statistics for network \( y \),
- \( \theta \) is the \( p \)-dimensional vector of coefficients for those statistics, and
- \( k(\theta) \) represents the quantity in the numerator summed over all possible networks (typically constrained to be all networks with the same node set as \( y \)).

In particular, the model implies that the probability attached to a network \( y \) only depends on the network via the vector of statistics \( g(y) \). Among other things, this means that maximum likelihood estimation may be carried out even if we don’t observe the network itself, as long as we know the observed value of \( g(y) \). We will see an example of this procedure in Section 2 of this tutorial.

The model statistics \( g(y) \): ERGM terms

The statistics \( g(y) \) can be thought of as the “covariates” in the model. In the network modeling context, these represent network features like density, homophily, triads, etc. In one sense, they are like covariates you might use in other statistical models. But they are different in one important respect: these \( g(y) \) statistics are functions of the network itself—each is defined by the frequency of a specific configuration of dyads observed in the network—so they are not measured by a question you include in a survey (e.g., the income of a node), but instead need to be computed on the specific network you have, after you have collected the data.

As a result, every term in an ERGM must have an associated algorithm for computing its value for your network. The `ergm` package in `statnet` includes about 150 term-computing algorithms. You can get an up-to-date list of all available terms, and the syntax for using them, by typing `?ergmTerm`. When using RStudio, it is possible to press the tab key after starting a line with `?ergm` to view the wide range of possible help options beginning with the letters `ergm`.

To obtain help for a specific term, use either `help("[name]-ergmTerm")` or the shorthand version `ergmTerm?[name]`, where `[name]` is the name of the term.
One key categorization of model terms is worth keeping in mind: terms are either dyad independent or dyad dependent. Dyad independent terms (like nodal homophily terms) imply no dependence between dyads—the presence or absence of a tie may depend on nodal attributes, but not on the state of other ties. Dyad dependent terms (like degree terms, or triad terms), by contrast, imply dependence between dyads. Dyad dependent terms have very different effects, and much of what is different about network models comes from these terms. They introduce complex cascading effects that can often lead to counter-intuitive and highly non-linear outcomes. In addition, a model with at least one dyad dependent term requires a different estimation algorithm, so when we use these terms below you will see some different components in the output.

ERGM probabilities at the tie level

The ERGM expression for the probability of the entire graph shown above can be re-expressed in terms of the conditional log-odds (that is, the logit of the conditional probability) of a single tie between two actors:

$$\logit P(Y_{ij} = 1 | y_{ij}^c) = \theta^\top \delta_{ij}(y),$$

where

- $Y_{ij}$ is the random variable for the state of the actor pair $i, j$ (with realization $y_{ij}$), and
- $y_{ij}^c$ signifies the complement of $y_{ij}$, i.e. the entire network $y$ except for $y_{ij}$.
- $\delta_{ij}(y)$ is a vector of the “change statistics” for each model term. The change statistic records how the $g(y)$ term changes if the $y_{ij}$ tie is toggled from off to on while fixing the rest of the network. So

$$\delta_{ij}(y) = g(y_{ij}^+) - g(y_{ij}^-),$$

where

- $y_{ij}^+$ is defined as $y_{ij}^c$ along with $y_{ij}$ set to 1, and
- $y_{ij}^-$ is defined as $y_{ij}^c$ along with $y_{ij}$ set to 0.

So $\delta_{ij}(y)$ equals the value of $g(y)$ when $y_{ij} = 1$ minus the value of $g(y)$ when $y_{ij} = 0$, but all other dyads are as in $y$. When this vector of change statistics is multiplied by the vector of coefficients $\theta$, the equation above shows that this dot product is the log-odds of the tie between $i$ and $j$, conditional on all other dyads remaining the same.

In other words, for an individual statistic, its change value for $Y_{ij}$ times its corresponding coefficient can be interpreted as that term’s contribution to the log-odds of that tie, conditional on all other dyads remaining the same.

2. Sample space constraints

Many applications take the sample space $\mathcal{Y}$ to be the power set $2^\mathcal{Y}$ of (possibly a subset of) all potential relationships. Yet it is sometimes desirable to restrict the sample space by placing constraints on which relationships $(i,j)$ are allowed in $\mathcal{Y}$ and further which networks $y \in 2^\mathcal{Y}$ are allowed in $\mathcal{Y}$.

For example, a bipartite network allows only edges connecting nodes from one subset, or mode, to nodes from its complement. Alternatively, we may wish to allow edges only within subsets of the node set, a situation often called a block-diagonal constraint. As still another, some applications impose a cap on the degree of any node, which constrains the sample space to include only those networks in which every node has a permitted degree.

Correct statistical inference for ERGMs depends on correctly incorporating constraints into the fitting process. They are specified using the constraints argument, a one-sided formula whose terms specify the constraints on the sample space.

For example, suppose we wish to constrain the sample space to only those networks with a particular edge density. To accomplish this, constraints = ~ edges specifies $\mathcal{Y}^{\text{edges}}(y) = \{y' \in \mathcal{Y} : |y'| = |y|\}$, where $y$
is the observed network specified on the left-hand side. Here is a simple example in which we generate a
random network according to a fitted ERGM and conditional on fixing the number of edges:

```r
data("faux.mesa.high")
fmh.fit <- ergm(faux.mesa.high ~ edges + nodematch("Grade"))
newnw <- simulate(fmh.fit, constraints = ~edges, nsim = 10)
```

<table>
<thead>
<tr>
<th></th>
<th>faux.mesa.high</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>203</td>
</tr>
</tbody>
</table>

The number of edges in the randomly-simulated `newnw` above will always be the same as in `faux.mesa.high`
due to the constraint.

A full list of currently implemented constraints is obtained via `?ergmConstraint`, and a specific constraint
called `[name]` can be looked up with `help("[name]-ergmConstraint")` or `ergmConstraint?[name]`. The
handling of various constraints by MCMC proposals in the `ergm` package is addressed in Krivitsky et al
(2022b).

**Dyad-independent constraints via the Dyads operator**

*Dyad-independent* constraints, which affect $Y$ only through $Y$ and do not induce stochastic dependencies
among the dyad states, may be combined arbitrarily because they do not require special Metropolis–Hastings
proposal algorithms for efficient sampling.

In the remainder of this section, we illustrate some of `ergm`'s constraints using a dataset due to Coleman
(1964). These data are self-reported friendship ties among 73 boys measured at two time points during the
1957–1958 academic year. They are included as a $2 \times 73 \times 73$ array and documented in the `sna` package.

Here, we use the Coleman data to create a `network` object with $2 \times 73$ nodes:

```r
library("sna")
data("coleman")
cole <- matrix(0, 2 * 73, 2 * 73)
# Upper left block:
cole[1:73, 1:73] <- coleman[1, , ]
# Lower right block:
cole[73 + (1:73), 73 + (1:73)] <- coleman[2, , ]
coleNW <- network(cole)
coleNW %v% "Semester" <- rep(c("Fall", "Spring"), each = 73)
coleNW
```

Network attributes:
- vertices = 146
- directed = TRUE
- hyper = FALSE
- loops = FALSE
multiple = FALSE
bipartite = FALSE
total edges= 506
  missing edges= 0
  non-missing edges= 506

Vertex attribute names:
  Semester vertex.names

No edge attributes

By construction, the coleNW network includes the Fall 1957 semester data and the Spring 1958 data as the upper left $73 \times 73$ and lower right $73 \times 73$ blocks, respectively. To see this structure, we create a simple function to plot an binary adjacency matrix so that the zeros and ones are different colors:

```r
BinaryMatrixPlot <- function(x) {
  n <- dim(x)[1]
  plot(0:n, 0:n, type="n", xaxt="n", yaxt="n", xlab="", ylab="", bty="n", ylim=c(n,0))
  therows <- row(x)[x>0]
  thecols <- col(x)[x>0]
  rect(thecols-1, therows-1, thecols, therows)
  rect(0,0,n,n)
}
```

Now we can visualize the constrained structure of the coleNW network, where the upper left corner depicts the relationships measured in the fall and the lower right corner depicts the spring relationships. Relationships are impossible in the upper right and lower left, a fact we must take into account when calculating statistical estimates.

```r
BinaryMatrixPlot(as.matrix(coleNW))
```

The Dyads(fix=NULL, vary=NULL) operator takes one or two ergm formulas that may contain only dyad-independent terms. For the terms in the fix= formula, dyads that affect the network statistic (i.e., have nonzero change statistic) for any the terms will be fixed at their current values. For the terms in the vary= formula, only those that change at least one of the terms will be allowed to vary, and all others will be fixed. A formula passed without an argument name will default to fix=, for consistency with other constraints’ semantics.

The key to our treatment of the coleNW network using the Dyads operator is the Semester vertex attribute:
In particular, the `nodematch("Semester")` term has a change statistic equal to one for exactly those dyads representing boys measured during the same semester, and this change statistic is zero otherwise. Therefore, in our 146-node directed network there are $146 \times 145 = 21,170$, total dyads, of which $2 \times 73 \times 72 = 10,512$, have nonzero change statistics for `nodematch("Semester")`.

We can easily see exactly how many total edges there are and verify that they all match on the “Semester” variable:

```r
summary(coleNW ~ edges + nodematch("Semester"))
```

<table>
<thead>
<tr>
<th></th>
<th>edges</th>
<th>nodematch.Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>506</td>
<td>506</td>
</tr>
</tbody>
</table>

Another way to verify that no edges exist between Fall semester and Spring semester nodes uses the `mm` (for mixingmatrix) term:

```r
summary(coleNW ~ mm("Semester", levels2 = TRUE))
```

<table>
<thead>
<tr>
<th></th>
<th>mm[Semester=Fall,Semester=Fall]</th>
<th>mm[Semester=Spring,Semester=Fall]</th>
<th>mm[Semester=Fall,Semester=Spring]</th>
<th>mm[Semester=Spring,Semester=Spring]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>243</td>
<td>0</td>
<td>0</td>
<td>263</td>
</tr>
</tbody>
</table>

If we ignore the constraints entirely, the `edges` coefficient is the log-odds, or logit, of $506 / 21170$:

```r
logit <- function (p) log(p / (1-p))
cbind(logit(506 / 21170),
      coef(ergm(coleNW ~ edges)))
```

<table>
<thead>
<tr>
<th></th>
<th>[,1]</th>
<th>[,2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-3.709612</td>
<td>-3.709612</td>
</tr>
</tbody>
</table>

Next, we use the `Dyads` constraint allowing only those dyads with nonzero change statistics for `nodematch("Semester")` to vary, and verify that we now obtain the log-odds of $506 / 10512$:

```r
cbind(logit(506 / 10512),
      coef(ergm(coleNW ~ edges,
                 constraints = ~ Dyads(vary = ~ nodematch("Semester")))))
```

<table>
<thead>
<tr>
<th></th>
<th>[,1]</th>
<th>[,2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-2.984404</td>
<td>-2.984404</td>
</tr>
</tbody>
</table>

A significant limitation of this particular constraint is that its initialization requires testing every possible dyad and therefore takes up time and memory in proportion to the square of the number of nodes.

**Constraints via blocks**

We may reproduce the example above using the `blocks` operator, which constrains changes to any dyads that involve certain pairs of categories defined by a particular nodal covariate. First, consider the full complement of statistics produced by the `nodemix` model term:

```r
summary(coleNW ~ nodemix("Semester",
                         levels = TRUE, levels2 = TRUE))
```
mix.Semester.Fall.Fall  mix.Semester.Spring.Fall  243  0
mix.Semester.Fall.Spring  mix.Semester.Spring.Spring  0  263

The `levels = TRUE` argument ensures that nodemix considers every value of "group" in constructing a mixing matrix of possible dyad combinations. The `levels2 = TRUE` argument ensures that, from the full complement of such possible combinations, every one is included as a statistic. By default, `levels = TRUE` whereas `levels2 = -1` since we frequently want to exclude at least one possible mixing combination to avoid collinearity in a model that also includes the `edges` term.

We may now use `levels2` in conjunction with `blocks` to select exactly which of the `nodemix` combinations should be constrained as fixed, namely, the second and third statistics from the full mixing matrix:

```r
# This coefficient estimate should match the example above
coeff(ergm(coleNW ~ edges,
          constraints = ~ blocks("Semester", levels2 = c(2, 3))))
edges
-2.984404
```

Additional examples using `levels2`, among other nodal attribute features, are contained in the `nodal_attributes` vignette within the `ergm` package.

### 3. Tuning ERGM estimation

Recent versions of `ergm` allow for more control over various aspects of the ERGM fitting process, such as the Metropolis-Hastings proposal distribution and various algorithmic control parameters. Here, we illustrate some of these features using an example from Section 5 of Krivitsky et al (2022) in which we estimate the parameters of an `ergm` for a hypothetical network on 50,000 nodes.

Our network will be based on the `cohab` dataset in the `ergm` package, which consists of three R objects, none of them a network object. These objects are based on aggregated statistics from the National Survey of Family Growth (NSFG).

- `cohab_PopWts`: A set of NSFG demographic attributes—sex, age, and race/ethnicity/immigration status—along with weights that have been adjusted to match the demographics of King County in Washington State.
- `cohab_TargetStats`: A vector of expected statistics for a 15-term ERGM applied to a network of 50,000 nodes, with in which a tie represents a heterosexual cohabition relationship between two nodes.
- `cohab_MixMat`: A Mixing matrix on the ‘race’ variable, giving the number of cohabiting male-female pairs of each possible combination of ‘race’ values. The five values of this variable, which captures some aspects of race/ethnicity/immigration status, are Black, Black immigrant, Hispanic, Hispanic immigrant, and White. The $5 \times 5$ mixing matrix is, like `cohab_PopWts`, based on NSFG data reweighted to match King County demographics.

Additional information about the `cohab` dataset is available by typing `help(cohab)`.

The column names of `cohab_PopWts` tell us which nodal attributes we need to set for our 50,000-node network:

```r
data("cohab")
names(cohab_PopWts)

[1] "weight"  "age"    "race"    "sex"    "sex.ident"
[6] "othr.net.deg" "agesq" "sqrt.age.adj"
```

The first column of `cohab_PopWts` is the set of weights, proportional to the probabilities we will use to construct our random set of 50,000 nodes so that it matches the demographics of King County. The value
of $\text{agesq}$ is simply the square of $\text{age}$, and $\text{sqrt.age.adj}$ is the square root of $\text{age}$ with a small upward adjustment for females.

Letting $\text{nw}$ denote our 50,000 network, the ERGM formula we wish to use is the same one used to generate the $\text{cohab\_TargetStats}$ object according to $\text{\texttt{?cohab}}$:

```r
cohab.formula <-
  (nw ~ edges + nodefactor("sex.ident", levels = 3)
    + nodecov("age") + nodecov("agesq")
    + nodefactor("race", levels = -5)
    + nodefactor("othr.net.deg", levels = -1)
    + nodematch("race", diff = TRUE) + absdiff("sqrt.age.adj"))
```

Recall that we do not actually need to observe the $\text{nw}$ network to fit an ERGM, although we do need the nodes in the network to be specified. Thus, our code first creates a network containing no edges in which the nodal attributes are selected according to the weights specified by $\text{cohab\_PopWts}$.

```r
set.seed(4321)
net_size <- 50000
nw <- network.initialize(net_size, directed = FALSE)
inds <- sample(seq_len(NROW(cohab_PopWts)), net_size, TRUE, cohab_PopWts$weight)
set.vertex.attribute(nw, names(cohab_PopWts)[-1], cohab_PopWts[inds,-1])
```

In this network, a tie is by definition a heterosexual cohabitation relationship and we assume that no individual may have more than one such relationship. We wish to constrain the set of allowable networks according these assumptions, which we may accomplish using the $\text{bd}$ (bounded degree) and $\text{blocks}$ constraints: Thus, the following formula will be passed to the $\text{ergm}$ function via the $\text{constraints}$= argument:

```r
constraint.formula <-
  (~ bd(maxout = 1)
    + blocks(attr = -sex, levels2 = diag(TRUE, 2)))
```

We may now fit the model. This takes a few minutes, so we’ll start it and describe some of the control parameters as it runs:

```r
set.seed(1234)
fit <- ergm(cohab.formula,
            target.stats = cohab_TargetStats,
            eval.loglik = FALSE,
            constraints = constraint.formula,
            control = snctrl(  
              MCMC.prop = ~strat(attr = ~race, empirical = TRUE) + sparse,  
              init.method = "MPLE",  
              init.MPLE.samplesize = 5e7,  
              MPLE.constraints.ignore = TRUE,  
              MCMLE.effectiveSize = 64,  
              SAN.nsteps = 5e7,  
              SAN.prop =~strat(attr = ~race, pmat = cohab_MixMat) + sparse))
```

In RStudio, typing $\text{?snctrl}$ shows all of the control arguments that may be used. The improvements to the fitting algorithm are not merely due to an expanded list of control arguments; coding efficiencies too have resulted in quicker fitting. According to Krivitsky et al (2022), the example we just considered took anywhere from 1.3 hours to 22.64 hours to fit using $\text{ergm}$ version 3.10, depending on the settings used.

Finally, we may examine the set of estimated coefficients.
**coef(summary(fit))**

<table>
<thead>
<tr>
<th>Term</th>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>z value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>0.785277529</td>
<td>0.4839693005</td>
<td>0</td>
<td>1.6225772</td>
</tr>
<tr>
<td>nodefactor.sex.ident.msmf</td>
<td>-1.509318824</td>
<td>0.2043290557</td>
<td>0</td>
<td>-7.3867068</td>
</tr>
<tr>
<td>nodecov.age</td>
<td>-0.401146744</td>
<td>0.0143411971</td>
<td>0</td>
<td>-27.9716360</td>
</tr>
<tr>
<td>nodecov.agesq</td>
<td>0.006836269</td>
<td>0.0002244601</td>
<td>0</td>
<td>30.4564931</td>
</tr>
<tr>
<td>nodefactor.race.B</td>
<td>1.398141315</td>
<td>0.154859095</td>
<td>0</td>
<td>8.9364176</td>
</tr>
<tr>
<td>nodefactor.race.BI</td>
<td>1.303563897</td>
<td>0.1645346127</td>
<td>0</td>
<td>7.9227336</td>
</tr>
<tr>
<td>nodefactor.race.H</td>
<td>2.748098886</td>
<td>0.1416603826</td>
<td>0</td>
<td>19.3992054</td>
</tr>
<tr>
<td>nodefactor.race.HI</td>
<td>1.430449585</td>
<td>0.1051492120</td>
<td>0</td>
<td>13.6039972</td>
</tr>
<tr>
<td>nodefactor.othr.net.deg.1</td>
<td>-4.542462604</td>
<td>0.1292577459</td>
<td>0</td>
<td>-35.1426723</td>
</tr>
<tr>
<td>nodematch.race.B</td>
<td>3.258740036</td>
<td>0.2650856025</td>
<td>0</td>
<td>12.7750841</td>
</tr>
<tr>
<td>nodematch.race.BI</td>
<td>3.914819910</td>
<td>0.2603546548</td>
<td>0</td>
<td>15.0364953</td>
</tr>
<tr>
<td>nodematch.race.H</td>
<td>0.050088295</td>
<td>0.1347766112</td>
<td>0</td>
<td>22.2815964</td>
</tr>
<tr>
<td>nodematch.race.HI</td>
<td>2.883540053</td>
<td>0.1490205662</td>
<td>0</td>
<td>19.349970</td>
</tr>
<tr>
<td>nodematch.race.W</td>
<td>3.404495855</td>
<td>0.1051492120</td>
<td>0</td>
<td>13.6039972</td>
</tr>
<tr>
<td>absdiff.sqrt.age.adj</td>
<td>-3.143347800</td>
<td>0.0571064224</td>
<td>0</td>
<td>-55.0436828</td>
</tr>
<tr>
<td></td>
<td>Pr(&gt;</td>
<td>z</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>edges</td>
<td>1.046798e-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.sex.ident.msmf</td>
<td>1.505100e-13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodecov.age</td>
<td>3.597251e-172</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodecov.agesq</td>
<td>9.827730e-204</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.race.B</td>
<td>4.085994e-19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.race.BI</td>
<td>2.323451e-15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.race.H</td>
<td>7.837344e-84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.race.HI</td>
<td>3.791330e-42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodefactor.othr.net.deg.1</td>
<td>1.504186e-270</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.race.B</td>
<td>2.259033e-37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.race.BI</td>
<td>4.233798e-51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.race.H</td>
<td>8.020641e-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.race.HI</td>
<td>2.040520e-83</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nodematch.race.W</td>
<td>5.75361e-110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absdiff.sqrt.age.adj</td>
<td>0.000000e+00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Term operators

**ergm** 4 introduces a new way to augment an **ergm** function call that we call a **term operator**, or simply **operator**. In mathematics, an operator is a function, like differentiation, that takes functions as its inputs; analogously, a term operator takes one or more ERGM formulas as input and transforms them by modifying their inputs and/or outputs.

Most **ergm** operators have the form **X(formula, ...)** where **X** is the name of the operator, typically capitalized, **formula** is a one-sided formula specifying the network statistics to be evaluated, and the remaining arguments control the transformation applied to the network before **formula** is evaluated and/or to the transformation applied to the network statistics obtained by evaluating **formula**. Operators are documented alongside other terms, accessible as **help("[name]-ergmTerm")** or **ergmTerm?[name]**

Filtering edges

The operator **F(formula, filter)** evaluates the terms in **formula** on a filtered network, with filtering specified by **filter**. Here, **filter** is the right-hand side of a formula that must contain one binary dyad-independent **ergm** term, having exactly one statistic with a dyadwise contribution of 0 for a 0-valued dyad.
That is, the term must be expressible as

$$g(y) = \sum_{(i,j) \in Y} f_{i,j}(y_{i,j}),$$  

where for all possible \((i,j)\), \(f_{i,j}(0) = 0\). One may verify that this condition implies that an ERGM containing the single term \(g(y)\) has the property that the dyads \(Y_{i,j}\) are jointly independent, which is why such a term is called “dyad-independent”. Examples of such terms include \texttt{nodemix}, \texttt{nodematch}, \texttt{nodefactor}, \texttt{nodecov}, and \texttt{edgecov}. Then, \texttt{formula} will be evaluated on a network constructed by taking \(y\) and keeping only those edges for which \(f_{i,j}(y_{i,j}) \neq 0\). This predicate can be modified slightly by very simple comparison or logical expressions in the \texttt{filter} formula. In particular, placing \(!\) in front of the term negates it (i.e., keep \((i,j)\) only if \(f_{i,j}(y_{i,j}) = 0\)) and comparison operators (==, <, etc.) allow comparing \(f_{i,j}(y_{i,j})\) to values other than 0.

Sampson’s Monks can provide illustrative examples. \texttt{ergm} includes a version of these data reporting cumulative liking nominations over the three time periods Sampson asked a group of monks to identify those they liked. We may plot this directed, 18-node network, where in the labels we use “L”, “O”, or “T” to indicate Sampson’s categorization of each monk as a Loyalist, Outcast, or Young Turk:

```r
set.seed(123)
data("sampson")
lab <- paste0(1:18, " ", substr(samplike $"group", 1, 1), ": ", samplike $"vertex.names")
plot(samplike, displaylabels=TRUE, label = lab)
```

As an example of the \texttt{F} operator, the code below uses four different methods to summarize the number of ties between pairs of nodes in the \texttt{Turks} group:

```r
summary(
  (samplike ~ nodematch("group", diff = TRUE, levels = "Turks")
  + F(- nodematch("group"), - nodefactor("group", levels = "Turks"))
  + F(- edges, - nodefactor("group", levels = "Turks") == 2)
  + F(- edges, - !nodefactor(~group != "Turks")))
)
```

Here, the third method works because this particular \(f_{i,j}(y_{i,j})\) counts how many of the two nodes \(i\) and \(j\) are Turks, and so equals 2 if and only if both are; and the fourth method works because the new \(f_{i,j}(y_{i,j})\) is 0 exactly when neither \(i\) nor \(j\) is a Turk.
It is also possible to filter on a quantitative variable. For instance, an alternative way to count the number of edges in `faux.mesa.high` that match on "Grade" is to report total edges after filtering by node pairs whose absolute difference on the "Grade" variable is less than 1:

```r
cbind(summary(faux.mesa.high ~ nodematch("Grade")),
      summary(faux.mesa.high ~ F(~ edges, ~ absdiff("Grade") < 1)))
```

```
[,1] [,2]
nodematch.Grade 163 163
```

While `filter` must be dyad-independent, `formula` can have dyad-dependent terms as well. For instance, we may count the transitive triples—i.e., triples \((i,j,k)\) where \(y_{i,j} = y_{j,k} = y_{i,k} = 1\)—in the `samplike` network, then perform the same count on the subnetwork consisting only of those edges connecting two monks not in attendance in the minor seminary of Cloisterville before coming to the monastery:

```r
summary((samplike ~ ttriple + F(~ ttriple, ~ nodefactor("cloisterville") == 0)))
```

```r
ttriple 154
F(nodefactor("cloisterville") == 0) ~ ttriple 12
```

**Treating directed networks as undirected**

The operator `Symmetrize(formula, rule)` evaluates the terms in `formula` on an undirected network constructed by symmetrizing the underlying directed network according to `rule`. The possible values of `rule`, which match the terminology of the `symmetrize` function of the `sna` package, are (a) “weak”, (b) “strong”, (c) “upper”, and (d) “lower”. These four values result in an undirected tie between \(i\) and \(j\), given any \(i < j\), if and only if:

- weak: either \(y_{i,j}\) or \(y_{j,i}\) equals 1;
- strong: both \(y_{i,j}\) and \(y_{j,i}\) equal 1;
- upper: \(y_{i,j} = 1\);
- lower: \(y_{j,i} = 1\)

For example, we can compute the number of node pairs \(i < j\) with reciprocated edges, equivalent to mutuality:

```r
summary((samplike ~ Symmetrize(- edges, "strong") + mutual))
```

```
Symmetrize(strong)-edges mutual
28 28
```

We can also compute the number of node pairs in which at least one edge is present; adding this value to the number of mutually connected node pairs (so that the latter are counted twice each) yields the total number of directed edges:

```r
summary((samplike ~ Symmetrize(- edges, "weak") + edges))
```

```
Symmetrize(weak)-edges edges
60 88
```

**Extracting subgraphs**

The operator `S(formula, attrs)` evaluates the terms in `formula` on an induced subgraph constructed from vertices identified by `attrs`, a formula that can be one-sided or, to make the induced subgraph bipartite,
two-sided. For instance, suppose that we wish to model the density and mutuality dynamics within the group “Young Turks” as different from those of the rest of the network:

\[
\text{coef(summary(}
\text{ergm(}
\text{(samplike ~ edges + mutual}
\text{ + S( ~ edges + mutual, ~ (group == "Turks"))),}
\text{control = snctrl(seed = 123))})
\]

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| edges -2.007074 | 0.2377493 | 0 | -8.441979 | 3.120095e-17 |
| mutual 2.351613 | 0.4996500 | 0 | 4.706519 | 2.519819e-06 |
| \(S((\text{group}==\text{"Turks"}))\)~edges 2.812378 | 0.8650057 | 0 | 3.251282 | 1.148857e-03 |
| \(S((\text{group}==\text{"Turks"}))\)~mutual -2.165222 | 1.1965294 | 0 | -1.809585 | 7.036009e-02 |

Thus, the density within the group is statistically significantly higher, whereas the reciprocation within the group is lower, though not statistically significantly at the 5% level.

The \text{attrs} argument of the \(S(\text{formula, attrs})\) operator either takes a value as explained in the \text{?nodal_attributes} online help or, to obtain a bipartite network, a two-sided formula with the left-hand side specifying the tails and the right-hand side specifying the heads.

As another example, consider the directed edges from non-Young Turks to Young Turks. Creating the induced subgraph from these edges results in a bipartite network, shown in the figure.

![Bipartite network](image)

The emphasized edges in the figure are those involved in a 4-cycle. We can define the bipartite network, and count the 4-cycles in two ways: If only directed edges from non-Turks to Turks (black in the figure) are viewed as bipartite edges, there are three 4-cycles.

\[
\text{summary((samplike ~}
\text{S( ~ cycle(4), (group != "Turks") ~ (group == "Turks"))))}
\]

\(S((\text{group}!=\text{"Turks"}), (\text{group}==\text{"Turks"}))\)~cycle4

3

If we also include edges from Turks to non-Turks (dotted red in the figure) via \text{Symmetrize} and the “weak” rule, we get two more.

\[
\text{summary(samplike}
\text{ ~ Symmetrize(}
\text{ ~ S( ~ cycle(4), (group != "Turks") ~ (group == "Turks")),}
\text{"weak")}
\]

\text{Symmetrize(weak)~S((group!="Turks"),(group=="Turks"))~cycle4}

5

The last example also illustrates that term operators may be nested arbitrarily.

Finally, we illustrate a common use case in which \text{Symmetrize} is used to analyze mutuality in a directed network as a function of a predictor. The \text{faux.dixon.high} dataset is a directed friendship network of
seventh through twelfth graders. Suppose we wish to check how strongly the tendency toward mutuality in friendships is affected by students’ closeness in grade level.

data("faux.dixon.high")
FDHfit <- ergm(faux.dixon.high ~ edges + mutual + absdiff("grade")
+ Symmetrize(~absdiff("grade"), "strong"),
control = snctrl(seed=321))
coef(summary(FDHfit))

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>z value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-3.2468082</td>
<td>0.05110162</td>
<td>0</td>
</tr>
<tr>
<td>mutual</td>
<td>3.2407587</td>
<td>0.12095858</td>
<td>0</td>
</tr>
<tr>
<td>absdiff.grade</td>
<td>-0.9145735</td>
<td>0.04309196</td>
<td>0</td>
</tr>
<tr>
<td>Symmetrize(strong)-absdiff.grade</td>
<td>-0.4237874</td>
<td>0.18035755</td>
<td>0</td>
</tr>
</tbody>
</table>

Pr(>|z|)
edges 0.000000e+00
mutual 3.972740e-158
absdiff.grade 5.762326e-100
Symmetrize(strong)-absdiff.grade 1.878819e-02

After correcting for the overall network density, the propensity for friendships to be reciprocated, and the predictive effect of grade difference on friendship formation, the difference in grade level has a statistically significant negative effect on the tendency to form mutual friendships (p-value = 0.019).

**Interaction effects**

For binary ERGMs, interactions between dyad-independent `ergm` terms can be specified in a manner similar to `lm` and `glm` via the : and * operators.

Let us first consider the colon (:) operator. Generally, if term \( A \) creates \( p_A \) statistics and term \( B \) creates \( p_B \) statistics, then \( A:B \) will create \( p_A \times p_B \) new statistics. If \( A \) and \( B \) are dyad-independent terms, expressed for \( a = 1, \ldots, p_A \) and \( b = 1, \ldots, p_B \) as

\[
\begin{align*}
g_a(y) & = \sum_{(i,j) \in V} u_{i,j}^a y_{i,j} \\
h_b(y) & = \sum_{(i,j) \in V} v_{i,j}^b y_{i,j}
\end{align*}
\]

for appropriate covariate matrices \( U^a \) and \( V^b \), then the corresponding interaction term is

\[
g_{a:b}(y) = \sum_{(i,j) \in V} u_{i,j}^a v_{i,j}^b y_{i,j}
\]

(2)

As an example, consider the Grade and Sex effects, expressed as model terms via `nodefactor`, in the `faux.mesa.high` dataset:

`summary(faux.mesa.high ~ nodefactor("Grade", levels = TRUE) : nodefactor("Sex"))`

nodefactor.Grade.7:nodefactor.Sex.M nodefactor.Grade.8:nodefactor.Sex.M
70 99
63 46
38 26

In the call above, we deliberately include all Grade-factor levels via `levels=TRUE`, whereas we employ the default behavior of `nodefactor` for the Sex factor, which leaves out one level. Thus, the 6-level Grade factor and the 2-level Sex factor, with one level of the latter omitted, produce \( 6 \times 1 \) interaction terms in this example.

The * operator, by contrast, produces all interactions in addition to the main effects or statistics. Therefore, in the scenario described above, \( A*B \) will add \( p_A + p_B + p_A \times p_B \) statistics to the model. Below, we use the
default behavior of `nodefactor` on both the 6-level Grade factor and the 2-level Sex factor, together with an additional `edges` term, to produce a model with $1 + 5 + 1 + 5 \times 1$ terms:

```r
m <- ergm(faux.mesa.high ~ edges + nodefactor("Grade") * nodefactor("Sex"))
print(summary(m, digits = 3))
```

Call:
`ergm(formula = faux.mesa.high ~ edges + nodefactor("Grade") * nodefactor("Sex"))`

Maximum Likelihood Results:

<table>
<thead>
<tr>
<th>Term</th>
<th>Estimate</th>
<th>Std. Error</th>
<th>MCMC %</th>
<th>z value</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>-3.028</td>
<td>0.173</td>
<td>0</td>
<td>-17.53</td>
</tr>
<tr>
<td>nodefactor.Grade.8</td>
<td>-1.424</td>
<td>0.263</td>
<td>0</td>
<td>-5.41</td>
</tr>
<tr>
<td>nodefactor.Grade.9</td>
<td>-1.166</td>
<td>0.229</td>
<td>0</td>
<td>-5.10</td>
</tr>
<tr>
<td>nodefactor.Grade.10</td>
<td>-1.633</td>
<td>0.357</td>
<td>0</td>
<td>-4.58</td>
</tr>
<tr>
<td>nodefactor.Grade.11</td>
<td>-0.328</td>
<td>0.237</td>
<td>0</td>
<td>-1.38</td>
</tr>
<tr>
<td>nodefactor.Grade.12</td>
<td>-0.794</td>
<td>0.324</td>
<td>0</td>
<td>-2.45</td>
</tr>
<tr>
<td>nodefactor.Sex.M</td>
<td>-1.764</td>
<td>0.240</td>
<td>0</td>
<td>-7.36</td>
</tr>
<tr>
<td>nodefactor.Grade.8:nodefactor.Sex.M</td>
<td>1.386</td>
<td>0.202</td>
<td>0</td>
<td>6.86</td>
</tr>
<tr>
<td>nodefactor.Grade.9:nodefactor.Sex.M</td>
<td>1.012</td>
<td>0.211</td>
<td>0</td>
<td>4.79</td>
</tr>
<tr>
<td>nodefactor.Grade.10:nodefactor.Sex.M</td>
<td>1.347</td>
<td>0.264</td>
<td>0</td>
<td>5.11</td>
</tr>
<tr>
<td>nodefactor.Grade.11:nodefactor.Sex.M</td>
<td>0.419</td>
<td>0.240</td>
<td>0</td>
<td>1.75</td>
</tr>
<tr>
<td>nodefactor.Grade.12:nodefactor.Sex.M</td>
<td>1.059</td>
<td>0.290</td>
<td>0</td>
<td>3.65</td>
</tr>
</tbody>
</table>

Pr(>|z|)

<table>
<thead>
<tr>
<th>Term</th>
<th>Pr(&gt;z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>edges</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.8</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.9</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.10</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.11</td>
<td>0.16714</td>
</tr>
<tr>
<td>nodefactor.Grade.12</td>
<td>0.01429 *</td>
</tr>
<tr>
<td>nodefactor.Sex.M</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.8:nodefactor.Sex.M</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.9:nodefactor.Sex.M</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.10:nodefactor.Sex.M</td>
<td>&lt; 1e-04 ***</td>
</tr>
<tr>
<td>nodefactor.Grade.11:nodefactor.Sex.M</td>
<td>0.08074 .</td>
</tr>
<tr>
<td>nodefactor.Grade.12:nodefactor.Sex.M</td>
<td>0.00026 ***</td>
</tr>
</tbody>
</table>

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 28987 on 20910 degrees of freedom
Residual Deviance: 2189 on 20898 degrees of freedom

AIC: 2213 BIC: 2308 (Smaller is better. MC Std. Err. = 0)

Interactions involving dyad-dependent terms are not straightforward, so ergm’s default behavior if : or * is used with a dyad-dependent term is to return an error. This default behavior may be changed by setting the `interact.dependent` option; see `help("ergm-options")` for more details.

Since interactions are defined by multiplying change statistics dyadwise and then summing over all dyads, interactions of terms are not necessarily the same as terms derived from products. Here is an example using the undirected Florentine marriage dataset:

```r
data("florentine")
summary(flomarriage ~ nodecov("wealth") : nodecov("wealth"))
```
The reason the two statistics above are not the same is that \( \text{nodecov("wealth")} \) is expressed as
\[
g(y) = \sum_{(i,j) \in Y} u_{i,j} y_{i,j}
\]
by setting \( u_{i,j} = \text{wealth}_i + \text{wealth}_j \). Thus, in the interaction term, each \( y_{i,j} \) is multiplied by \((\text{wealth}_i + \text{wealth}_j)^2\), whereas in the \text{wealth}^2 term, each \( y_{i,j} \) is multiplied by only \( \text{wealth}_i^2 + \text{wealth}_j^2 \).

Reparametrizing the model

The term operator \( \text{Sum(formulas, label)} \) allows arbitrary linear combinations of existing statistics to be added to the model. Suppose \( g_1(y), \ldots, g_K(y) \) is a set of \( K \) vector-valued network statistics, each corresponding to one or more \text{ergm} terms and of arbitrary dimension. Also suppose that \( A_1, \ldots, A_K \) is a set of known constant matrices all having the same number of rows such that each matrix multiplication \( A_k g_k(y) \) is well-defined. Then it is now possible to define the statistic
\[
g_{\text{Sum}}(y) = \sum_{k=1}^K A_k g_k(y).
\]

The first argument to \( \text{Sum} \) is a formula or a list of \( K \) formulas, each representing a vector statistic. If a formula has a left-hand side, the left-hand side will be used to define the corresponding \( A_k \) matrix: If it is a scalar or a vector, \( A_k \) will be a diagonal matrix thus multiplying each element by its corresponding element; and if it is a matrix, \( A_k \) will be used directly. When no left-hand side is given, \( A_k \) is defined as 1. To simplify this function for some common cases, if the left-hand side is "sum" or "mean", the sum (or mean) of the statistics in the formula is calculated.

As an example, consider a vector of statistics consisting of the numbers of friendship ties received by each subgroup of Sampson’s monks:

```r
summary(samplike ~ nodeifactor("group", levels = TRUE))
```

<table>
<thead>
<tr>
<th>nodeifactor.group.Loyal</th>
<th>nodeifactor.group.Outcasts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>13</td>
</tr>
<tr>
<td>nodeifactor.group.Turks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>46</td>
</tr>
</tbody>
</table>

We may create a single statistic equal to the friendship ties received by both groups of non-Outcasts by adding the first and third components of the \text{nodefactor} vector, either by left-multiplying by \([1 \ 0 \ 1]\) or by deselecting the second component at the \text{nodefactor} level and summing the remaining two:

```r
summary(samplike ~
  Sum(cbind(1, 0, 1) - nodeifactor("group", levels = TRUE), "nf.L_T")
  + Sum("sum" ~ nodeifactor("group", levels = -2), "nf.L_T"))
```

<table>
<thead>
<tr>
<th>Sum-nf.L_T</th>
<th>Sum-nf.L_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>75</td>
</tr>
</tbody>
</table>

Whereas the \( \text{Sum} \) operator calculates linear combinations of network statistics, the \( \text{Prod} \) operator calculates the products of their powers. As of this writing, it is implemented for nonnegative statistics only, by first applying the \( \text{Log} \) operator (which returns the natural logarithm, \text{log} in R, of the statistics passed to it), then the \( \text{Sum} \) operator, and finally the \( \text{Exp} \) operator (which returns the exponential function, \text{exp} in R). As a simple illustration, we may verify that the \( \text{Sum} \) and \( \text{Prod} \) operators do in fact produce network statistics as expected if we simply use each with a list of formulas having no left hand side:
summary(faux.dixon.high - edges + mutual
+ Sum(list(~ edges, ~ mutual), "EdgesAndMutual")
+ Prod(list(~ edges, ~ mutual), "EdgesAndMutual"))

<table>
<thead>
<tr>
<th>edges</th>
<th>mutual</th>
<th>Sum-EdgesAndMutual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1197</td>
<td>219</td>
<td>1416</td>
</tr>
</tbody>
</table>

Exp-Sum-EdgesAndMutual
262143

The `For()` operator can be used to parametrize other terms. This operator evaluates the formula given to it substituting the specified loop counter variable with each element in a sequence. Provide:

1. An `ergm` formula in an unnamed argument containing placeholders to be substituted.
2. One or more named arguments in the form `<placeholder> = <sequence>`, indicating what to substitute:
   - vector or list
   - LHS formula or function evaluated on network

The following are equivalent ways to compute differential homophily:

```r
data(sampson)
(groups <- sort(unique(samplike$v"group"))) # Sorted list of groups.

[1] "Loyal" "Outcasts" "Turks"
```

The “normal” way:

```r
summary(samplike ~ nodematch("group", diff=TRUE))

<table>
<thead>
<tr>
<th>nodematch.group.Loyal</th>
<th>nodematch.group.Outcasts</th>
<th>nodematch.group.Turks</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
```

One element at a time, specifying a list:

```r
summary(samplike ~ For(~nodematch("group", levels=., diff=TRUE),
. = groups))

<table>
<thead>
<tr>
<th>nodematch.group.Loyal</th>
<th>nodematch.group.Outcasts</th>
<th>nodematch.group.Turks</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
```

One element at a time, specifying a function that returns a list:

```r
summary(samplike ~ For(~nodematch("group", levels=., diff=TRUE),
. = function(nw) sort(unique(nw$v"group"))))

<table>
<thead>
<tr>
<th>nodematch.group.Loyal</th>
<th>nodematch.group.Outcasts</th>
<th>nodematch.group.Turks</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
```

One element at a time, specifying a formula whose RHS expression returns a list:

```r
summary(samplike ~ For(~nodematch("group", levels=., diff=TRUE),
. = ~sort(unique(.v"group"))))

<table>
<thead>
<tr>
<th>nodematch.group.Loyal</th>
<th>nodematch.group.Outcasts</th>
<th>nodematch.group.Turks</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
```

Here, `absdiff()` is being computed for each combination of attribute and power, the “normal” way:

```r
data(florentine)
summary(flomarriage ~ absdiff("wealth", pow=1) + absdiff("priorates", pow=1) +
```
5. Modeling multiple networks

The ergm.multi package provides support for modeling multiple networks, as we illustrate in this section using datasets supplied in that package.

First, load the ergm.multi package along with some helper packages. You might need to use install.packages first if any of these is not yet installed:

```r
library(ergm.multi)
library(dplyr)
library(purrr)
library(tibble)
```

Obtaining data

The list of networks studied by Goeyvaerts et al (2018) is included in this package:

```r
data(Goeyvaerts)
length(Goeyvaerts)
```

[1] 318

An explanation of the networks, including a list of their network (%n%) and vertex (%v%) attributes, can be obtained via ?Goeyvaerts. A total of 318 complete networks were collected, then two more excluded due to “nonstandard” family composition:

```r
Goeyvaerts %>% discard(~%n%, "included") %>% map(as_tibble, unit="vertices")
```

[[1]]
# A tibble: 4 x 5
vertex.names age gender na role
  <int> <int> <chr> <lgl> <chr>
1    1    32 F FALSE Mother
2    2    48 F FALSE Grandmother
3    3    32 M FALSE Father
4    4    10 F FALSE Child

[[2]]
# A tibble: 3 x 5
vertex.names age gender na role
  <int> <int> <chr> <lgl> <chr>
1    1    32 F FALSE Mother
2    2    48 F FALSE Grandmother
3    3    32 M FALSE Father
To reproduce the analysis, exclude them as well:

\[
G \leftarrow \text{Goeyvaerts} \%>% \text{keep}(\text{"included"})
\]

Data summaries

Obtain weekday indicator, network size, and density for each network, and summarize them as in Goeyvaerts et al (2018), Table 1:

\[
G \%>% \text{map}(-\text{list}(\text{weekday} = ., \text{"weekday"}), \\
\quad n = \text{network.size}(.), \\
\quad d = \text{network.density}(.,)) \%>% \text{bind_rows}() \%>% \\
\quad \text{group_by}(\text{weekday, } n = \text{cut}(n, c(1,2,3,4,5,9))) \%>% \\
\quad \text{summarize}(\text{nnets} = n(), \ p1 = \text{mean}(d==1), \ m = \text{mean}(d)) \%>% \text{kable}()
\]

<table>
<thead>
<tr>
<th>weekday</th>
<th>n</th>
<th>nnets</th>
<th>p1</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>(1,2]</td>
<td>3</td>
<td>1.0000000</td>
<td>1.0000000</td>
</tr>
<tr>
<td>FALSE</td>
<td>(2,3]</td>
<td>19</td>
<td>0.7368421</td>
<td>0.8771930</td>
</tr>
<tr>
<td>FALSE</td>
<td>(3,4]</td>
<td>48</td>
<td>0.8541667</td>
<td>0.9618056</td>
</tr>
<tr>
<td>FALSE</td>
<td>(4,5]</td>
<td>18</td>
<td>0.7777778</td>
<td>0.9500000</td>
</tr>
<tr>
<td>FALSE</td>
<td>(5,9]</td>
<td>3</td>
<td>1.0000000</td>
<td>1.0000000</td>
</tr>
<tr>
<td>TRUE</td>
<td>(1,2]</td>
<td>9</td>
<td>1.0000000</td>
<td>1.0000000</td>
</tr>
<tr>
<td>TRUE</td>
<td>(2,3]</td>
<td>53</td>
<td>0.9056604</td>
<td>0.9622642</td>
</tr>
<tr>
<td>TRUE</td>
<td>(3,4]</td>
<td>111</td>
<td>0.7747748</td>
<td>0.9279279</td>
</tr>
<tr>
<td>TRUE</td>
<td>(4,5]</td>
<td>39</td>
<td>0.6410256</td>
<td>0.8974359</td>
</tr>
<tr>
<td>TRUE</td>
<td>(5,9]</td>
<td>13</td>
<td>0.4615385</td>
<td>0.8454212</td>
</tr>
</tbody>
</table>

Reproducing ERGM fits

We now reproduce the ERGM fits. First, we extract the weekday networks:

\[
G.\text{wd} \leftarrow G \%>% \text{keep}(\text{"weekday"})
\]

\[
\text{length}(G.\text{wd})
\]

\[1\] 225

Next, we specify the multi-network model using the \text{N(formula, lm)} operator. This operator will evaluate the \text{ergm} \text{formula} on each network, weighted by the predictors passed in the one-sided \text{lm} \text{formula}, which is interpreted the same way as that passed to the built-in \text{lm()} function, with its “\text{data}” being the table of network attributes.

Since different networks may have different compositions, to have a consistent model, we specify a consistent list of family roles.

\[
\text{roleset} \leftarrow \text{sort}(\text{unique}(\text{unlist}(\text{lapply}(G.\text{wd}, \text{"role"}))))
\]

We now construct the formula object, which will be passed directly to \text{ergm()}:

\[
\text{# Networks()} \text{ function tells ergm()} \text{ to model these networks jointly.}
\]

\[
\text{f.wd} \leftarrow \text{Networks(G.\text{wd})} - \\
\text{# This N()} \text{ operator adds three edge counts:} \\
\text{\text{N(-edges,}} \\
\text{\quad -# one total for all networks (intercept implicit as in lm),}
\]

\[ I(n \leq 3) + \text{# one total for only small households, and} \]
\[ I(n \geq 5) \text{ # one total for only large households.} \]

# This \( N() \) construct evaluates each of its terms on each network, 
# then sums each statistic over the networks:
\[ N(\text{# First, mixing statistics among household roles, including only} \] 
\[ \text{# father-mother, father-child, and mother-child counts.} \] 
\[ \text{# Since tail < head in an undirected network, in the} \] 
\[ \text{# levels2 specification, it is important that tail levels (rows) } \] 
\[ \text{# come before head levels (columns). In this case, since} \] 
\[ \text{# "Child" < "Father" < "Mother" in alphabetical order, the} \] 
\[ \text{# row= and col= categories must be sorted accordingly.} \] 
\~mm("role", levels = I(roleset), 
\text{levels2=.-%in%list(list(row="Father",col="Mother"),} 
\text{list(row="Child",col="Father"),} 
\text{list(row="Child",col="Mother")))) + \]

# Second, the nodal covariate effect of age, but only for 
# edges between children.
\text{F(-nodecov("age"), -nodematch("role", levels=I("Child"))) +} 

# Third, 2-stars.
\text{kstar(2)} 

\[ N(\text{# This \( N() \) adds one triangle count, totalled over all households} \] 
\[ \text{# with at least 6 members.} \] 
\[ N(-\text{triangles, -I(n>6)}) \]

See `ergmTerm?mm` for documentation on the \textit{mm} term used above. Now, we can fit the model:

```r
fit.wd <- ergm(f.wd)
```

```r
summary(fit.wd)
```

Call:
\text{ergm(formula = f.wd)}

Monte Carlo Maximum Likelihood Results:

| Term                                      | Estimate | Std. Error | MCMC % z value | Pr(>|z|) |
|-------------------------------------------|----------|------------|----------------|----------|
| N(1)-edges                                | 0.84277  | 0.53315    | 0              | 0.113936 |
| N(I(n <= 3)TRUE)-edges                    | 1.48525  | 0.43480    | 0              | 0.20811  |
| N(I(n >= 5)TRUE)-edges                    | -0.80143 | 0.20811    | -0.8          | 0.000636 |
| N(1)-mm[role=Child,role=Father]           | -0.63765 | 0.48484    | -0.6          | 0.000118 |
| N(1)-mm[role=Child,role=Mother]           | 0.13614  | 0.52994    | 2             | 0.000118 |
| N(1)-mm[role=Father,role=Mother]          | 0.25377  | 0.58752    | 0             | 0.000118 |
| N(1)-F(nodematch("role", levels=I("Child"))-nodecov.age | -0.07154 | 0.01695    | -0.0          | 0.21225  |
| N(1)-kstar2                               | -0.25767 | 0.21225    | -0.2          | 0.21225  |
| N(1)-triangle                             | 2.05168  | 0.31374    | 2             | 0.21225  |
| N(I(n > 6)TRUE)-triangle                  | -0.28102 | 0.10850    | -0.2          | 0.21225  |

0.53315 0.48484 0.52994 0.58752 0.01695 0.21225 0.31374 0.10850
Similarly, we can extract the weekend network, and fit it to a smaller model. We only need one N() operator, since all statistics are applied to the same set of networks, namely, all of them.

G.we <- G %>% discard(~n, "weekday")
fit.we <- ergm(Networks(G.we) ~ N(~edges + mm("role", levels = I(roleset),
levels2 = ~. %in% list(list(row="Father", col="Mother"),
list(row="Child", col="Father"), list(row="Child", col="Mother"))) +
F(~nodecov("age"), ~nodematch("role", levels = I("Child"))) +
kstar(2) +
triangles))

summary(fit.we)

Call:
ergm(formula = Networks(G.we) ~ N(~edges + mm("role", levels = I(roleset),
levels2 = ~. %in% list(list(row="Father", col="Mother"),
list(row="Child", col="Father"), list(row="Child", col="Mother"))) +
F(~nodecov("age"), ~nodematch("role", levels = I("Child"))) +
kstar(2) +
triangles))

Monte Carlo Maximum Likelihood Results:

<table>
<thead>
<tr>
<th>Term</th>
<th>Estimate</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>N(1)-edges</td>
<td>2.1574</td>
<td>1.4273</td>
</tr>
<tr>
<td>N(1)-mm[role=Child,role=Father]</td>
<td>-1.1795</td>
<td>1.4697</td>
</tr>
<tr>
<td>N(1)-mm[role=Child,role=Mother]</td>
<td>0.1391</td>
<td>1.5732</td>
</tr>
<tr>
<td>N(1)-mm[role=Father,role=Mother]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N(1)-F(nodematch(&quot;role&quot;,levels=I(&quot;Child&quot;)))~nodecov.age</td>
<td>-4.220</td>
<td>&lt; 1e-04</td>
</tr>
<tr>
<td>N(1)-kstar2</td>
<td>0.224754</td>
<td></td>
</tr>
<tr>
<td>N(1)-triangle</td>
<td>0.009600</td>
<td></td>
</tr>
<tr>
<td>N(I(n &gt;= 6)TRUE)-triangle</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
N(1)~mm[role=Father,role=Mother]  -0.7507  1.5343
N(1)~F(nodematch("role",levels=I("Child"))-nodecov.age  -0.1765  0.0519
N(1)~kstar2  -0.8488  0.3592
N(1)~triangle  3.5582  0.7602

MCMC % z value Pr(>|z|)
N(1)~edges 0 1.511 0.130662
N(1)~mm[role=Child,role=Father] 0 -0.803 0.422257
N(1)~mm[role=Child,role=Mother] 0 0.088 0.929571
N(1)~mm[role=Father,role=Mother] 0 -0.489 0.624649
N(1)~F(nodematch("role",levels=I("Child"))-nodecov.age 0 -3.400 0.000674
N(1)~kstar2 0 -2.363 0.018127
N(1)~triangle 0 4.680 < 1e-04

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 802.7 on 579 degrees of freedom
Residual Deviance: 133.2 on 572 degrees of freedom

AIC: 147.2 BIC: 177.8 (Smaller is better. MC Std. Err. = 0.7987)

Diagnostics
Perform diagnostic simulation (Krivitsky et al, 2022c), summarize the residuals, and make residuals vs. fitted and scale-location plots:
gof.wd <- gofN(fit.wd, GOF = ~ edges + kstar(2) + triangles)

Constructing simulation model(s).
Constructing GOF model.
Simulating unconstrained sample.
Collating the simulations.
Summarizing.

summary(gof.wd)

`$`Observed/Imputed values`
edges    kstar2    triangle
Min. : 1.000    Min. : 0.00    Min. : 0.000
1st Qu.: 3.000    1st Qu.: 3.00    1st Qu.: 1.000
Median : 6.000    Median :12.00    Median : 4.000
Mean : 5.778    Mean :13.55    Mean : 4.324
3rd Qu.: 6.000    3rd Qu.:12.00    3rd Qu.: 4.000
Max. :18.000    Max. :78.00    Max. :23.000
NA's :9    NA's :9

$`Fitted values`
edges  kstar2  triangle
Min. : 0.860  Min. : 2.620  Min. : 0.700
1st Qu.: 2.950 1st Qu.: 7.622 1st Qu.: 2.348
Median : 5.570 Median :10.610 Median : 3.390
Mean : 5.754 Mean :13.409 Mean : 4.277
3rd Qu.: 5.840 3rd Qu.:11.473 3rd Qu.: 3.743
NA's :9 NA's :9

Pearson residuals
edges kstar2 triangle
Min. :-5.498132 Min. :-4.2209 Min. :-4.20054
1st Qu.: 0.215984 1st Qu.: 0.2233 1st Qu.: 0.19607
Median : 0.364671 Median : 0.3825 Median : 0.39214
Mean : 0.000977 Mean : 0.0052 Mean : 0.01461
3rd Qu.: 0.477977 3rd Qu.: 0.5123 3rd Qu.: 0.54272
Max. : 1.752754 Max. : 2.0089 Max. : 1.93910
NA's :9 NA's :9

Variance of Pearson residuals
edges [1] 1.153315
kstar2 [1] 1.091188
triangle [1] 1.028553

Std. dev. of Pearson residuals
edges [1] 1.073925
kstar2 [1] 1.044599
triangle [1] 1.014176

Variances of Pearson residuals substantially greater than 1 suggest unaccounted-for heterogeneity.

library(ggplot2)
autoplot(gof.wd)

Loading required namespace: ggrepel

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
[[2]]

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
\[ \sqrt{\text{Std. Pearson resid.}} \]

\begin{verbatim}
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
\end{verbatim}
```
geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
The plots don’t look unreasonable.

Also make plots of residuals vs. square root of fitted and vs. network size:

```R
autoplot(gof.wd, against=sqrt(.fitted))
```

[[1]]

'geom_smooth()' using method = 'loess' and formula = 'y ~ x'
[[2]]

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
\$\sqrt{\text{Std. Pearson resid.}}\$

\[\text{geom_smooth()}\]

\$\text{using method = 'loess' and formula = 'y ~ x'}\$

Warning: Removed 9 rows containing non-finite outside the scale range (\text{stat_smooth()}).

Warning: Removed 9 rows containing missing values or values outside the scale range (\text{geom_point()}).

Warning: Removed 9 rows containing missing values or values outside the scale range (\text{geom_text_repel()}).
```
geom_smooth() using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range.

Warning: Removed 9 rows containing missing values or values outside the scale range.

Warning: Removed 9 rows containing missing values or values outside the scale range.
```

[[4]]
\[
\sqrt{\text{Std. Pearson resid.}}
\]

\[
\text{sqrt('Fitted values')}
\]

[[5]]

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'

Warning: Removed 9 rows containing non-finite outside the scale range (`stat_smooth()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_point()`).

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
autoplot(gof.wd, against=factor(n))

[[1]]
Warning: Computation failed in `stat_boxplot()`.
Caused by error in `loadNamespace()`:  
! there is no package called 'quantreg'
[[2]]
Warning: Computation failed in `stat_boxplot()`.
Caused by error in `loadNamespace()`:
! there is no package called 'quantreg'
Warning: Removed 9 rows containing non-finite outside the scale range (\texttt{stat_boxplot()}).

Warning: Computation failed in \texttt{stat_boxplot()}.  
Caused by error in \texttt{loadNamespace()}: 
! there is no package called 'quantreg'

Warning: Removed 9 rows containing missing values or values outside the scale range (\texttt{geom_text_repel()}).
Warning: Removed 9 rows containing non-finite outside the scale range (`stat_boxplot()`).

Warning: Computation failed in `stat_boxplot()`.
Caused by error in `loadNamespace()`:
! there is no package called 'quantreg'

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
[[5]]
Warning: Removed 9 rows containing non-finite outside the scale range (`stat_boxplot()`).
Warning: Computation failed in `stat_boxplot()`.
Caused by error in `loadNamespace()`:
  ! there is no package called 'quantreg'
Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
Warning: Removed 9 rows containing non-finite outside the scale range (`stat_boxplot()`).

Warning: Computation failed in `stat_boxplot()`.
Caused by error in `loadNamespace()`:
  ! there is no package called 'quantreg'

Warning: Removed 9 rows containing missing values or values outside the scale range (`geom_text_repel()`).
It looks like network-size effects are probably accounted for.

6. Estimation in the presence of missing data

It is quite common that network data are incomplete in various ways. The `ergm` package includes the capability to handle missing edge data, whereas other types of missingness such as missing nodal information are not addressed.

We illustrate using the `samplike` dataset used earlier. Consider a simple model with edges, mutuality (reciprocated dyads), transitive ties, and cyclical ties. For the sake of comparison, we first fit the model assuming no missing edge data. First, verify that `samplike` has no missing edges:

```r
print(samplike)
```

Network attributes:
- vertices = 18
- directed = TRUE
- hyper = FALSE
- loops = FALSE
- multiple = FALSE
- total edges= 88
  - missing edges= 0
  - non-missing edges= 88

Vertex attribute names:
- cloisterville group vertex.names

Edge attribute names:
nominations
Now fit the model:

```
summary(full.fit <-
  ergm(samplike ~ edges + mutual + transitiveties + cyclicalties,
       eval.loglik = TRUE), control = snctrl(seed = 321))
```

Call:
```
ergm(formula = samplike ~ edges + mutual + transitiveties + cyclicalties,
      eval.loglik = TRUE)
```

Monte Carlo Maximum Likelihood Results:

|        | Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|--------|----------|------------|--------|---------|----------|
| edges  | -1.8942  | 0.3648     | 0      | -5.192  | <1e-04 ***|
| mutual | 2.4790   | 0.4375     | 0      | 5.666   | <1e-04 ***|
| transitiveties | 0.5225 | 0.3076     | 0      | 1.699   | 0.0894 .   |
| cyclicalties   | -0.4546 | 0.2493     | 0      | -1.823  | 0.0682 .   |

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 424.2 on 306 degrees of freedom
Residual Deviance: 329.5 on 302 degrees of freedom

AIC: 337.5 BIC: 352.4 (Smaller is better. MC Std. Err. = 0.4547)

Next, suppose that Monk #1 (John Bosco) refused to respond during all three waves, rendering his replies missing:

```
samplike1 <- samplike
samplike1[1, ] <- NA
print(samplike1)
```

Network attributes:
- vertices = 18
- directed = TRUE
- hyper = FALSE
- loops = FALSE
- multiple = FALSE
- total edges= 99
- missing edges= 17
- non-missing edges= 82

Vertex attribute names:
- cloisterville group vertex.names

Edge attribute names:
- nominations

If we pass this modified object to `ergm`, it will automatically calculate the MLE under the assumption that the monk’s refusal is unrelated to his choice of relations, i.e., that the data are ignorably missing with respect to the specified model:

```
summary(m1.fit <-
  ergm(samplike1 ~ edges + mutual + transitiveties + cyclicalties,
       eval.loglik = TRUE), control = snctrl(seed = 321))
```
Call:
\[ \text{ergm(formula = samplike1 ~ edges + mutual + transitiveties + cyclicalties, eval.loglik = TRUE)} \]

Monte Carlo Maximum Likelihood Results:

|              | Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|--------------|----------|------------|--------|---------|----------|
| edges        | -2.0219  | 0.3918     | 0      | -5.161  | <1e-04 *** |
| mutual       | 2.4080   | 0.4734     | 0      | 5.087   | <1e-04 *** |
| transitiveties | 0.4585  | 0.4021     | 0      | 1.140   | 0.254    |
| cyclicalties | -0.2777  | 0.3701     | 0      | -0.750  | 0.453    |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 400.6 on 289 degrees of freedom
Residual Deviance: 313.7 on 285 degrees of freedom

AIC: 321.7 BIC: 336.4 (Smaller is better. MC Std. Err. = 0.6715)

The degrees of freedom associated with the missing data fit have decreased because unobserved dyads do not carry information.

For the theoretical grounding of the \textit{ergm} package’s algorithm for networks with missing edges, see Krivitsky et al (2022d), which is based on the framework developed by Handcock and Gile (2010). For more on the ignorability assumption for edge variables, see Handcock and Gile (2010).

The estimation approach above can be extended to other types of incomplete network observation. Karwa et al. (2017) applied it to fit arbitrary ERGMs to networks whose dyad values had been stochastically perturbed—ties added and removed at random, with known probabilities—in order to preserve privacy. Another use case is multiple imputation for networks with missing data, in which multiple random versions of the full network are constructed by randomly inserting values for unobserved dyads according to probabilities that are determined based on, say, some type of logistic regression model.

These mechanisms may be invoked by passing an \texttt{obs.constraints} formula, specifying how the network of interest was observed. Of particular interest are the following constraints:

- \textit{observed} restricts the proposal to changing only those dyads that are recorded as missing.
- \textit{egocentric} \((\text{attr = NULL, direction = c("both", "out", "in")})\) restricts the proposal to changing only those dyads that would not be observed in an egocentric sample. That is, dyads cannot be modified that are incident on vertices for which attribute specification \texttt{attr} has value \texttt{TRUE} or, if \texttt{attr} is \texttt{NULL}, the vertex attribute "\texttt{na}" has value \texttt{FALSE}. For directed networks, \texttt{direction="out"} only preserves the out-dyads of those actors, and \texttt{direction="in"} preserves their in-dyads.
- \textit{dyadnoise} \((\texttt{p01, p10})\) Unlike the others, this is a soft constraint to adjust the sampled distribution for dyad-level noise with known perturbation probabilities, which can arise in a variety of contexts. It is assumed that the observed LHS network is a noisy observation of some unobserved true network, with \texttt{p01} giving the dyadwise probability of erroneously observing a tie where the true network had a non-tie and \texttt{p10} giving the dyadwise probability of erroneously observing a nontie where the true network had a tie. \texttt{p01} and \texttt{p10} can be either both be scalars or both be adjacency matrices of the same dimension as that of the LHS network giving these probabilities.

We may use the \texttt{obs.constraints} argument to re-fit the model above:

\begin{verbatim}
> samplike2 <- samplike
> samplike2[,1] <- 0 # Careful! Zeros are not the same as missing, but...
> samplike2 $\%\%$ "refused" <- rep(c(TRUE,FALSE),c(1,17)) # This new nodal covariate labels who is missing
\end{verbatim}
Network attributes:
- vertices = 18
- directed = TRUE
- hyper = FALSE
- loops = FALSE
- multiple = FALSE
- total edges = 82
  - missing edges = 0
  - non-missing edges = 82

Vertex attribute names:
- cloisterville group refused vertex.names

Edge attribute names:
- nominations

Call:
```R
m2.fit <-
ergm(samplike2 ~ edges + mutual + transitiveties + cyclicalties,
observerstraints = ~ egocentric(~ !refused, "out"),
control = snctrl(seed = 123))
```

Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| edges    | -2.0149    | 0.4104 | 0       | -4.910   | <1e-04 *** |
| mutual   | 2.4407     | 0.4792 | 0       | 5.093    | <1e-04 *** |
| transitiveties | 0.4519 | 0.4335 | 0       | 1.042    | 0.297 |
| cyclicalties | -0.2832 | 0.3903 | 0       | -0.726   | 0.468 |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 400.6 on 289 degrees of freedom
Residual Deviance: 314.0 on 285 degrees of freedom

AIC: 322 BIC: 336.6 (Smaller is better. MC Std. Err. = 0.5995)

Finally, the observational process can be included as a part of the network dataset using the `%ergmlhs%` operation. If we do this, we do not need to explicitly pass the `obs.constraints` argument to the `ergm` function:
```R
samplike2 %ergmlhs% "obs.constraints" <- ~ egocentric(~ !refused, "out")
summary(m3.fit <-
ergm(samplike2 ~ edges + mutual + transitiveties + cyclicalties),
control = snctrl(seed = 231))
```

Call:
```
ergm(formula = samplike2 ~ edges + mutual + transitiveties + cyclicalties)
```
Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| edges    | -2.0409    | 0.3773 | 0       | -5.409   | <1e-04 *** |
| mutual   | 2.4716     | 0.4719 | 0       | 5.238    | <1e-04 *** |
| transitiveities | 0.4148 | 0.4128 | 0       | 1.005    | 0.315     |
| cyclicalties | -0.2314 | 0.3691 | 0       | -0.627   | 0.531     |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 400.6 on 289 degrees of freedom
Residual Deviance: 313.1 on 285 degrees of freedom

AIC: 321.1 BIC: 335.8 (Smaller is better. MC Std. Err. = 0.6085)

7. Multilayer networks

Also known as multiplex, multirelational, or multivariate networks, in a multilayer network a pair of actors can have multiple simultaneous relations of different types. For example, in the famous Lazega lawyer, each pair of lawyers in the firm can have an advice relationship, a coworking relationship, a friendship relationship, or any combination thereof. Application of ERGMs to multilayer networks has a long history (e.g., Pattison and Wasserman 1999, Lazega and Pattison 1999), and a number of R packages exist for analysing and estimating them.

`ergm.multi` implements the general approach of Krivitsky, Koehly, and Marcum (2020) for specifying multilayer ERGMs, including Layer Logic and the various cross-layer specifications. Its advantages include seamless integration with `ergm()`, number of layers limited only by computing power, ability to incorporate any ERGM effects into the framework, handling of networks that have both directed and undirected layers, and experimental multimode/multilevel support.

Preparing multilayer networks for analysis

To keep things simple (and fast), we will use the Florentine dataset included with `ergm`. It comprises two networks of relations among Florentine families, one of marriages, the other of business relations. Although the following examples only include two layers, `ergm.multi` supports an arbitrary number.

```r
# Method 1: list of networks
deflo <- Layer(list(m = flomarriage, b = flobusiness))

# Method 2: networks as arguments
deflo <- Layer(m = flomarriage, b = flobusiness)

# Method 3: edge attributes:
deflo2 <- flomarriage | flobusiness # superset of all edges in any layer
# set attributes
deflo2[,, names.eval="m"] <- as.matrix(flomarriage)
deflo2[,, names.eval="b"] <- as.matrix(flobusiness)
deflo2
```

Network attributes:
- vertices = 16
- directed = FALSE
- hyper = FALSE
- loops = FALSE

45
multiple = FALSE
bipartite = FALSE
total edges= 27
   missing edges= 0
   non-missing edges= 27

Vertex attribute names:
   vertex.names

Edge attribute names:
   b m
flo <- Layer(flo2, c("m","b"))

flo

Combined 2 networks on '.LayerID'/'.LayerName':
1: n = 16, directed = FALSE, bipartite = FALSE, loops = FALSE
2: n = 16, directed = FALSE, bipartite = FALSE, loops = FALSE

Network attributes:
   vertices = 32
   directed = FALSE
   hyper = FALSE
   loops = FALSE
   multiple = FALSE
   bipartite = FALSE
   ergm:
      Length Class Mode
consstraints 2 formula call
   total edges= 35
      missing edges= 0
      non-missing edges= 35

Vertex attribute names:
   .bipartite .LayerID .LayerName .undirected vertex.names

Edge attribute names:
   b m
flo is now a network with some additional metadata set.

Incidentally, we can extract the network defined by a particular edge attribute using the network_view() helper function:

   all(as.matrix(network_view(flo2, "m")) == as.matrix(flomarriage))

[1] TRUE

We will also use the Lazega Lawyers data to illustrate available techniques. This dataset is included with the ergm.multi package:

data(Lazega)
Lazega

Network attributes:
   vertices = 71
   directed = TRUE
hyper = FALSE
loops = FALSE
multiple = FALSE
bipartite = FALSE
total edges= 1574
    missing edges= 0
    non-missing edges= 1574

Vertex attribute names:
      age gender office practice school seniority status vertex.names yrs_frm

Edge attribute names not shown

This network has a number of vertex attributes, including each lawyer's demographic information, status in the firm, and professional history. It also has three non-NA edge attributes,

```
list.edge.attributes(Lazega)
```

```
[1] "advice"   "coworker"  "friendship" "na"
```

```
head(Lazega %e%"advice")
```

```
[1] 1 1 1 1 1 0
```

```
head(Lazega %e%"coworker")
```

```
[1] 0 0 0 0 0 0
```

```
head(Lazega %e%"friendship")
```

```
[1] 0 0 0 0 0 1
```

each representing a type of relationship collected. Note that the edges of the network itself are a superset of all layers' edges:

```
all(Lazega %e%"advice" + Lazega %e%"coworker" + Lazega %e%"friendship" > 0)
```

```
[1] TRUE
```

We can extract the individual layers using a helper function `network_view()`:

```
(L.a <- network_view(Lazega, "advice"))
```

Network attributes:
   vertices = 71
   directed = TRUE
   hyper = FALSE
   loops = FALSE
   multiple = FALSE
   bipartite = FALSE
   total edges= 892
      missing edges= 0
      non-missing edges= 892

Vertex attribute names:
      age gender office practice school seniority status vertex.names yrs_frm

Edge attribute names not shown
L.c <- network_view(Lazega, "coworker")
L.f <- network_view(Lazega, "friendship")

library(sna)
# Plot the advice network
coord <- gplot(L.a,
  # Colour the vertex according to the practice.
  vertex.col=as.numeric(as.factor(Lazega%v%"practice")+1,
  # Size the vertex according the seniority (2 if associate, 3 if partner):
  vertex.cex=2+(Lazega%v%"status"=="partner"),
  # # sides according to gender (2+1 for Female, 2+2 for Male):
  vertex.sides=as.numeric(as.factor(Lazega%v%"gender")+2,
  displayisolates=FALSE)
  text(coord, label=substr(Lazega%v%"office",1,1))
# Also, add a legend for colour, shape, plotting symbol, and letter:
legend("topleft",legend=levels(as.factor(Lazega%v%"practice")),
  col=2:3,pch=19, title="Practice")
legend("topright",legend=levels(as.factor(Lazega%v%"gender")),
  pch=c(2,5), title="Gender")
legend("bottomright",legend=levels(factor(Lazega%v%"status")),
  pt.cex=c(2,3), pch=1, title="Status")
legend("bottomleft",legend=levels(factor(Lazega%v%"office")),
  pch=substr(levels(factor(Lazega%v%"office")),1,1),title="Location")

# Plot the coworker network
coord <- gplot(L.c,
  # Colour the vertex according to the practice.
  vertex.col=as.numeric(as.factor(Lazega%v%"practice")+1,
  # Size the vertex according the seniority (2 if associate, 3 if partner):
  vertex.cex=2+(Lazega%v%"status"=="partner"),
  # # sides according to gender (2+1 for Female, 2+2 for Male):
  vertex.sides=as.numeric(as.factor(Lazega%v%"gender")+2,
  displayisolates=FALSE)
  text(coord, label=substr(Lazega%v%"office",1,1))
# Also, add a legend for colour, shape, plotting symbol, and letter:
legend("topleft",legend=levels(as.factor(Lazega%v%"practice")),
  col=2:3,pch=19, title="Practice")
# Plot the friendship network
coord <- gplot(L.f,
# Colour the vertex according to the practice.
  vertex.col=as.numeric(as.factor(Lazega$"practice"))+1,
# Size the vertex according the seniority (2 if associate, 3 if partner):
  vertex.cex=2*(Lazega$"status"=="partner"),
# sides according to gender (2+1 for Female, 2+2 for Male):
  vertex.sides=as.numeric(as.factor(Lazega$"gender"))+2,
  displayisolates=FALSE)
text(coord, label=substr(Lazega$"office",1,1))
# Also, add a legend for colour, shape, plotting symbol, and letter:
legend("topleft",legend=levels(as.factor(Lazega$"practice")),
  col=2:3,pch=19, title="Practice")
legend("topright",legend=levels(as.factor(Lazega$"gender")),
  pch=c(2,5), title="Gender")
legend("bottomright",legend=levels(factor(Lazega$"status")),
  pt.cex=c(2,3), pch=1, title="Status")
legend("bottomleft",legend=levels(factor(Lazega$"office")),
  pch=substr(levels(factor(Lazega$"office")),1,1),title="Location")
To model it, we need to tell `ergm` to interpret it as a multilayer network, which we do using the `Layer()` function (not an ERGM term). In this case, the most convenient way to do so is by referencing edge attributes:

```r
LLazega <- Layer(Lazega, c("advice", "coworker", "friendship"))
```

However, an optionally named list of networks or networks as named arguments are also accepted. We will use it to shorten the layer names:

```r
LLazega <- Layer(a=L.a, c=L.c, f=L.f)
```

More generally, `Layer()` will produce the “least common denominator” network: if any unipartite layers are present, bipartite layers will be coerced to unipartite (with some additional metadata to ensure that disallowed edges are never formed), and if any directed layers are present, all layers will be coerced to directed. It is possible to use `Symmetrize()` and `S()` (Subgraph) operators to evaluate undirected and bipartite effects on these layers.

### Specifying models for multilayer networks

Given the `Layer()` construct on the LHS of an `ergm()` formula, we can use `layer-aware` terms on the RHS. By convention, layer-aware terms have capital \( L \) appended to them. For example, `mutualL` is a layer-aware generalization of `mutual`. These terms have one or more explicit (usually optional) layer specification arguments. By convention, an argument that requires one layer specification is named \( L= \) and one that requires a list of specifications (constructed by `list()` or just `c()`) is named \( Ls= \); and a specification of the form \(-.\) is a placeholder for all observed layers.

We can get a list of layer-aware terms currently visible to ERGM with

```r
search.ergmTerms(keywords="layer-aware")
```

Found 10 matching ergm terms:

- `CMBL(Ls=-.)` (binary)
  - Conway–Maxwell–Binomial dependence among layers

- `ddspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)` (binary)
  - Dyadwise shared partners on layers

- `dspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)` (binary)
  - Edgewise shared partners on layers
Layer Logic

But how do we specify layers?

Layer Logic is an approach to specifying layers, their transformations, and interactions between two or more layers by constructing logical layers, which are layers constructed by evaluating logical expressions on observed layers.

Each formula’s right-hand side describes an observed layer or some “logical” layer, whose ties are a function of corresponding ties in observed layers. (Krivitsky et al. 2020)

The observed layers can be referenced either by name or by number (i.e., order in which they were passed to Layer). When referencing by number, enclose the number in quotation marks (e.g., "1") or backticks (e.g., `1`).

Standard arithmetical, comparison, and logical operations can be used, as well as some mathematical functions such as `abs()`, `round()`, and `sign()`. Standard operator precedence is followed applies, so use of parentheses is recommended to ensure the logical expression is what it looks like.

For example, if LHS is `Layer(A=nwA, B=nwB)`, both `~`2` and ~B refer to `nwB`, while `A&B` refers to a logical layer that has ties that are in `nwA` but not in `nwB`.

Transpose function `t()` applied to a directed layer will reverse the direction of all relations (transposing the sociomatrix). Unlike the others, it can only be used on an observed layer directly. For example, `~t(`2`) is valid but ~t(1&2`) is not.

At this time, logical expressions that produce complete graphs from empty graph inputs (e.g., `A==B` or `!A`) are not supported.

The `L(formula, Ls) operator` This is probably the most frequently used layer-aware term: it takes an arbitrary binary `ergm` formula and evaluates it on each observed or logical layer specified in Ls, and then adds up the results elementwise. A very common usage is to write L(~[TERMS], ~.) to fit an effect of [TERMS] homogeneous over all layers.
However, note that while the statistics are the same, the MLEs are different:

```
summary(LLazega ~ L(edges, ~f))
L(f)-edges
575
summary(L.f-edges)
edges
575
summary(LLazega ~(edges, c(~f, ~a)))
L((f,a))-edges
1467
summary(L.f-edges) + summary(L.a-edges)
edges
1467
summary(LLazega ~(edges, ~.))
L(.)-edges
2571
summary(L.f-edges) + summary(L.c-edges) + summary(L.a-edges)
edges
2571
summary(LLazega ~(edges, ~f|a))
L(f|a)-edges
1109
summary((L.f|L.a)-edges)
edges
1109
```

Call:
```
ergm(Layer(f = L.f, a = L.a) ~ L(edges, ~f | a))
```
Monte Carlo Maximum Likelihood Coefficients:
```
L(f|a)-edges
-2.342
```

```
summary(LLazega ~(edges, -f|a))
```
```
Call:
ergm(formula = Layer(f = L.f, a = L.a) ~ L(edges, -f | a))
```
Monte Carlo Maximum Likelihood Coefficients:
```
L(f|a)-edges
-2.342
```

```
ergm((L.f|L.a)-edges)
```
```
Call:
ergm(formula = (L.f | L.a) ~ edges)
```
Maximum Likelihood Coefficients:
```
edges
-1.247
```
This is because the sample space for the first one is all-possible two-layer networks, whereas the sample space for the second one is all possible one-layer networks. Thus, the normalizing constant is different.

Modelling association between layers

- CMBL(Ls): **Conway–Maxwell-binomial distribution:** By “default”, if \( m \) layers in Ls are homogeneous, the number of the layers with an edge should be Binomial\((m, p)\) where \( p \) is the density of each layer. CMB replaces the \( \binom{m}{y_{i,j}} \) in the binomial density with \( \binom{m}{y_{i,j}}^{\theta_{CMB}} \); a positive coefficient implies positive dependence and a negative one induces negative dependence.

Is there an overall positive association between nominations?

```r
summary(ergm(LLazega - L(~edges, ~.) + CMBL(~.)))
```

Call:
```r
ergm(formula = LLazega ~ L(~edges, ~.) + CMBL(~.))
```

Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| L(.)~edges -0.82841 | 0.01878 | 0 | -44.12 | <1e-04 *** |
| CMBL(~.) 1.42167 | 0.03479 | 0 | 40.87 | <1e-04 *** |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 20670 on 14910 degrees of freedom
Residual Deviance: 12290 on 14908 degrees of freedom

AIC: 12294 BIC: 12309 (Smaller is better. MC Std. Err. = 3.847)

Yes!

What if we focus on giving advice rather than receiving?

```r
summary(ergm(LLazega - L(~edges, c(~t(a), ~c, ~f)) + CMBL(c(~t(a), ~c, ~f))))
```

Call:
```r
ergm(formula = LLazega ~ L(~edges, c(~t(a), ~c, ~f)) + CMBL(c(~t(a), ~c, ~f)))
```

Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| L((t(a),c,f))~edges -0.89666 | 0.02148 | 1 | -41.75 | <1e-04 *** |
| CMBL(list(~t(a),~c,~f)) 1.21809 | 0.03489 | 0 | 34.91 | <1e-04 *** |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 20670 on 14910 degrees of freedom
Residual Deviance: 12732 on 14908 degrees of freedom

AIC: 12736 BIC: 12751 (Smaller is better. MC Std. Err. = 4.932)

Still yes.

**Direct layer logic effects**  Consider a two-layer model for a dyad \((i, j)\),

\[
\Pr( Y_{i,j,A} = y_{i,j,A} \land Y_{i,j,B} = y_{i,j,B} ) \propto \exp(\theta_1 y_{i,j,A} + \theta_2 y_{i,j,B} + \theta_3 y_{i,j,A} \cap B),
\]
for some logical operation □.

What happens if we fix $y_{i,j,B}$ and look at the conditional probability of $\Pr(Y_{i,j,A} = 1)$?

- **$a \& b$: Conjunction**: $\logit^{-1}(\theta_1 + \theta_3 y_{i,j,B})$, so $\theta_3 > 0 \implies$ presence of an edge in layer $b$ will increase the probability of the edge in layer $a$.

- **xor$(a,b)$: Mutual exclusivity**: $\logit^{-1}(\theta_1 + \theta_3 (-1)^{y_{i,j,B}})$, so $\theta_3 > 0 \implies$ presence of an edge in layer $b$ will decrease the probability of the edge in layer $a$.

- **$a | b$: Substitutability**: $\logit^{-1}(\theta_1 + \theta_3 (1 - y_{i,j,B}))$, so $\theta_3 > 0 \implies$ presence of an edge in layer $b$ will undo a “default” increase in the probability of the edge in layer $a$.

**summary(ergm(Layer(a=L.a,c=L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~a & c)))**

Call:

```r
ergm(formula = Layer(a = L.a, c = L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~a & c))
```

Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| L(a)-edges | -2.43931 | 0.05655 | 0 | -43.14 | <1e-04 *** |
| L(c)-edges | -1.92166 | 0.04613 | 0 | -41.66 | <1e-04 *** |
| L(a&c)-edges | 2.54346 | 0.08299 | 0 | 30.65 | <1e-04 *** |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 13780 on 9940 degrees of freedom
Residual Deviance: 8951 on 9937 degrees of freedom

AIC: 8957  BIC: 8979  (Smaller is better. MC Std. Err. = 3.07)

**summary(ergm(Layer(a=L.a,c=L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~xor(a,c)))**

Call:

```r
ergm(formula = Layer(a = L.a, c = L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~xor(a,c)))
```

Monte Carlo Maximum Likelihood Results:

| Estimate | Std. Error | MCMC % | z value | Pr(>|z|) |
|----------|------------|--------|---------|----------|
| L(a)-edges | -1.16375 | 0.04221 | 0 | -27.57 | <1e-04 *** |
| L(c)-edges | -0.64265 | 0.04132 | 0 | -15.55 | <1e-04 *** |
| L(xor(a,c))-edges | -1.27716 | 0.04052 | 0 | -31.52 | <1e-04 *** |

---

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Null Deviance: 13780 on 9940 degrees of freedom
Residual Deviance: 8951 on 9937 degrees of freedom

AIC: 8957  BIC: 8978  (Smaller is better. MC Std. Err. = 3.54)

**summary(ergm(Layer(a=L.a,c=L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~a | c)))**

Call:

```r
ergm(formula = Layer(a = L.a, c = L.c) ~ L(~edges, ~a) + L(~edges, ~c) + L(~edges, ~a | c))
```
Monte Carlo Maximum Likelihood Results:

| Estimate  | Std. Error | MCMC % | z value | Pr(>|z|) |
|-----------|------------|--------|---------|----------|
| L(a)-edges| 0.11157    | 0.05773| 0       | 1.933    | 0.0533   |
| L(c)-edges| 0.63421    | 0.06919| 0       | 9.166    | <1e-04   ***|
| L(a|c)-edges| -2.55271   | 0.07640| 0       | -33.413  | <1e-04   ***|

---

Signif. codes:  0 '***'  0.001 '**'  0.01 '*'  0.05 '.'  0.1 ' ' 1

Null Deviance: 13780 on 9940 degrees of freedom
Residual Deviance: 8950 on 9937 degrees of freedom

AIC: 8956  BIC: 8978  (Smaller is better. MC Std. Err. = 2.848)

Cross-layer reciprocity

- **mutualL(Ls)**: *Layer-aware mutuality*: Given Ls=c(~a, ~b), counts the number of ordered pairs (i, j) for which y_{i,j, A} = 1 ∧ y_{j,i, B} = 1.
- **L(~..., ~a&t(b))**: *Reversal layer logic*: Another way of expressing the above.

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?

Net of within-layer density and mutuality and co-occurrence, does advice reciprocate coworking?
- `twostarL(Ls, type, distinct)`: **Cross-layer two-star or two-path**: `Ls` is a list of two layers of interest (e.g., `list(~a,~b)`), and `type` is
  - "any" Undirected: instances where node `i` has an edge in layer `a` and an edge in layer `b`. (Whether these edges may be coincident depends on `distinct`.)
  - "out"/"in" Directed: instances where node `i` has out-edge/in-edge in `a` and out-edge/in-edge in `b`. (Whether these edges may be coincident depends on `distinct`.)
  - "path" Directed: instances where `i` has in-edge in `a` and an out-edge in `b`. (Whether these edges may be reciprocal depends on `distinct`.)

For illustration purposes, I will set `distinct=FALSE`. In your case, it should be driven by substantive or theoretical considerations.

```
summary(LLazega~twostarL(c(~c,-f), type="out", distinct=FALSE))
twostarL(c<>f)
   10563
# Equivalently...
sum(   rowSums(as.matrix(L.c))  # Outdegree of each node in layer c   *   rowSums(as.matrix(L.f))  # Outdegree of each node in layer f )
[1] 10563
```

```
summary(LLazega~twostarL(c(~c,-f), type="in", distinct=FALSE))
twostarL(c<f)
   10118
# Equivalently...
sum(   colSums(as.matrix(L.c))  # Indegree of each node in layer c   *   colSums(as.matrix(L.f))  # Indegree of each node in layer f )
[1] 10118
```

```
summary(LLazega~twostarL(c(~c,-f), type="path", distinct=FALSE))
twostarL(c>>f)
   9890
# Equivalently...
sum(   colSums(as.matrix(L.c))  # Indegree of each node in layer c   *   rowSums(as.matrix(L.f))  # Outdegree of each node in layer f )
[1] 9890
```

**Multilayer triadic effects**  Triadic effects are generalized by `gwdspL`, `gwespL`, and `gwnspL` which take the standard arguments of `gw*sp` and `dgw*sp` effects, including `type`, and, in addition, `L.base`, `L.path`, and `L.in_order`, specifying the layer of the base (for `gwespL` and `gwnspL`), the layer of the two-path, and whether, for the directed case, the order of the two-path matters.
### References

For a general orientation to the **statnet** packages, the best place to start is the special volume of the *Journal of Statistical Software* (JSS) devoted to **statnet**: [https://www.jstatsoft.org/issue/view/v024](https://www.jstatsoft.org/issue/view/v024). The nine papers in this volume cover a wide range of theoretical and practical topics related to ERGMs, and their implementation in **statnet**. However, this volume was written in 2008. The **statnet** code base has evolved considerably since that time, and with the release of **ergm** version 4.0, the most current paper describing the capabilities of the **ergm** package is the following preprints:


For social scientists, a good introductory application paper is:


**Goodness of Fit**


**Temporal ERGMs**


Multiple and Multilayer ERGMS


Missing Data

