Package 'ergm.multi'

July 22, 2025

Version 0.3.0

Date 2025-06-14

Title Fit, Simulate and Diagnose Exponential-Family Models for Multiple or Multilayer Networks

Depends R (>= 4.2.0), ergm (>= 4.9.0), network (>= 1.19.0)

Imports statnet.common (>= 4.12.0), rlang (>= 1.1.6), purrr (>= 1.0.4), tibble (>= 3.3.0), glue (>= 1.8.0), rle (>= 0.10.0), Rdpack (>= 2.6.4), networkLite (>= 1.1.0), Matrix, methods, parallel

LinkingTo ergm

Suggests rmarkdown (>= 2.29), knitr (>= 1.50), dplyr (>= 1.1.4), testthat (>= 3.2.3), ggplot2 (>= 3.5.2), ggrepel (>= 0.9.6), Rglpk, generics (>= 0.1.4)

RdMacros Rdpack

BugReports https://github.com/statnet/ergm.multi/issues

Description A set of extensions for the 'ergm' package to fit multilayer/multiplex/multirelational networks and samples of multiple networks. 'ergm.multi' is a part of the Statnet suite of packages for network analysis. See Krivitsky, Koehly, and Marcum (2020) <doi:10.1007/s11336-020-09720-7> and Krivitsky, Coletti, and Hens (2023) <doi:10.1080/01621459.2023.2242627>.

License GPL-3 + file LICENSE

URL https://statnet.org

VignetteBuilder rmarkdown, knitr

RoxygenNote 7.3.2.9000

Config/testthat/parallel true

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

Author Pavel N. Krivitsky [aut, cre] (ORCID: <https://orcid.org/0000-0002-9101-3362>), Mark S. Handcock [ctb],

David R. Hunter [ctb],
Chad Klumb [ctb],
Pietro Coletti [ctb],
Joyce Cheng [ctb]

Maintainer Pavel N. Krivitsky <pavel@statnet.org>

Repository CRAN

Date/Publication 2025-06-15 02:20:06 UTC

Contents

`ergm.multi-package` *ergm.multi: Fit, Simulate and Diagnose Exponential-Family Models for Multiple or Multilayer Networks*

Description

A set of extensions for the 'ergm' package to fit multilayer/multiplex/multirelational networks and samples of multiple networks. 'ergm.multi' is a part of the Statnet suite of packages for network analysis. See Krivitsky, Koehly, and Marcum (2020) doi:10.1007/s11336020097207 and Krivitsky, Coletti, and Hens (2023) doi:10.1080/01621459.2023.2242627.

Multilayer network models

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 Lazega lawyer data set included with this package, 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 (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 et al. (2020) for specifying multilayer ERGMs, including Layer Logic and the various cross-layer specifications. Its features include:

seamless integration with `ergm()`: Multilayer specification is contained entirely in an `ergm()`-style formula and can be nested with any other `ergm()` terms, including dynamic and multi-network.

unlimited layers: The number of layers in the modeled network is limited only by computing power.

flexibility and simplicity: *Any* valid binary ERGM can be specified for any layer or a logical combination of layers using simple term operators.

heterogeneous layers: A network can have directed and undirected layers, which can be modeled jointly.

multimode/multilevel support (experimental): With some care, it is possible to specify models for unipartite and bipartie layers over different subsets of actors, which can be used to specify multimode models.

See `Layer()` and `ergmTerm?L` for examples.

Multi-network models

Joint modeling of independent samples of networks on disjoint sets of actors have a long history as well (Zijlstra et al. 2006, Slaughter and Koehly 2016, Stewart et al. 2019, and Vega Yon et al. 2021, for example). **ergm.multi** facilitates fixed-effect models for samples of networks (possibly heterogeneous in size and composition), using a multivariate linear model for each network's ERGM parameters, with network-level attributes serving as predictors, as formulated by Slaughter and Koehly (2016) and Krivitsky et al. (2023).

Its features include:

seamless integration with `ergm()`: Multi-network model specification is contained entirely in an `ergm()`-style formula and can be nested with any other `ergm()` terms, including dynamic and multilayer.

flexibility and simplicity: *Any* valid binary or valued ERGM can be specified for the networks, using simple term operators and the network-level specification with an `lm()`-style formula.

See `Networks()`, `ergmTerm?N` for specification, `gofN()` for diagnostic facilities, and `vignette("Goeyvaerts_reproductio` for a demonstration.

Author(s)

Maintainer: Pavel N. Krivitsky <pavel@statnet.org> (ORCID)

Other contributors:

- Mark S. Handcock <handcock@stat.ucla.edu> [contributor]
- David R. Hunter <dhunter@stat.psu.edu> [contributor]
- Chad Klumb <cklumb@gmail.com> [contributor]
- Pietro Coletti <pietro.coletti@uhasselt.be> [contributor]
- Joyce Cheng <joyce.cheng@student.unsw.edu.au> [contributor]

References

Krivitsky PN, Coletti P, Hens N (2023). "A Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks." *Journal of the American Statistical Association*, **118**(544), 2213-2224. doi:10.1080/01621459.2023.2242627.

Krivitsky PN, Koehly LM, Marcum CS (2020). "Exponential-family Random Graph Models for Multi-layer Networks." *Psychometrika*, **85**(3), 630–659. doi:10.1007/s11336020097207.

Lazega E, Pattison PE (1999). "Multiplexity, Generalized Exchange and Cooperation in Organizations: A Case Study." *Social Networks*, **21**(1), 67–90. doi:10.1016/S03788733(99)000027.

Pattison P, Wasserman S (1999). "Logit Models and Logistic Regressions for Social Networks: II. Multivariate Relations." *British Journal of Mathematical and Statistical Psychology*, **52**(2), 169–193.

Slaughter AJ, Koehly LM (2016). "Multilevel Models for Social Networks: Hierarchical Bayesian Approaches to Exponential Random Graph Modeling." *Social Networks*, **44**, 334–345. doi:10.1016/j.socnet.2015.11.002.

Stewart J, Schweinberger M, Bojanowski M, Morris M (2019). "Multilevel Network Data Facilitate Statistical Inference for Curved ERGMs with Geometrically Weighted Terms." *Social Networks*, **59**, 98–119. doi:10.1016/j.socnet.2018.11.003.

Vega Yon GG, Slaughter A, de la Haye K (2021). "Exponential Random Graph Models for Little Networks." *Social Networks*, **64**, 225–238. doi:10.1016/j.socnet.2020.07.005.

Zijlstra BJH, Van Duijn MAJ, Snijders TAB (2006). "The Multilevel p_2 Model: A Random Effects Model for the Analysis of Multiple Social Networks." *Methodology*, **2**(1), 42.

See Also

Useful links:

- https://statnet.org
- Report bugs at https://github.com/statnet/ergm.multi/issues

---

`as_tibble.combined_networks` *An* `as_tibble` *method for combined networks.*

Description

A method to obtain a network attribute table from a `combined_networks` object, falling back to the `network::as_tibble.network()` if vertex or edge attributes are required.

Usage

```
## S3 method for class 'combined_networks'
as_tibble(
  x,
  attrnames = (match.arg(unit) %in% c("vertices", "networks")),
  ...,
  unit = c("edges", "vertices", "networks"),
  .NetworkID = ".NetworkID",
  .NetworkName = ".NetworkName",
  store.nid = FALSE
)
```

Arguments

`x`	a `combined_networks` (inheriting from `network::network`).
`attrnames`	a list (or a selection index) for attributes to obtain; for combined networks, defaults to all.
`...`	additional arguments, currently passed to unlist()].
`unit`	whether to obtain edge, vertex, or network attributes.
`.NetworkID, .NetworkName`	Optional strings indicating the vertex attributes used to distinguish and name the networks; intended to be used by term developers.
`store.nid`	whether to include columns with network ID and network name; the columns will be named with the arguments passed to `.NetworkID` and `.NetworkName`.

See Also

`network::as_tibble.network()`

b1dspL-ergmTerm *Dyadwise shared partners for dyads in the first bipartition on layers*

Description

This term adds one network statistic to the model for each element in `d`; the ith such statistic equals the number of dyads in the first bipartition with exactly `d[i]` shared partners. (Those shared partners, of course, must be members of the second bipartition.) This term can only be used with bipartite networks.

Usage

```
# binary: b1dspL(d, Ls.path=NULL)
```

Arguments

`d`	a vector of distinct integers.
`Ls.path`, `L.in_order`	a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Note

This term does not support multilayer networks with heterogeneous bipartedness. This may change in the future.

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: bipartite, layer-aware, undirected, binary

b2dspL-ergmTerm *Dyadwise shared partners for dyads in the second bipartition on layers*

Description

This term adds one network statistic to the model for each element in `d` ; the i th such statistic equals the number of dyads in the second bipartition with exactly `d[i]` shared partners. (Those shared partners, of course, must be members of the first bipartition.) This term can only be used with bipartite networks.

Usage

```
# binary: b2dspL(d, Ls.path=NULL)
```

Arguments

`d`	a vector of distinct integers
`Ls.path`, `L.in_order`	a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Note

This term does not support multilayer networks with heterogeneous bipartedness. This may change in the future.

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: bipartite, layer-aware, undirected, binary

CMBL-ergmTerm *Conway–Maxwell-Binomial dependence among layers*

Description

Models marginal dependence layers within each dyad by imposing a Conway–Maxwell-Binomial (CMB) distribution on the number of layers in each dyad that have a tie.

The term adds one statistic to the model, equalling the sum over all the dyads in the network of $\log\{E!(R-E)!/R!\}$, where E is the number of layers in `Ls` with an edge in that dyad and R being the total number of layers in `Ls` .

Usage

```
# binary: CMBL(Ls=~.)
```

Arguments

`Ls`	a list (constructed by `list()` or `c()` of at least two Layer Logic specifications (c.f. Layer Logic section in the `Layer()` documentation).

Details

A positive coefficient induces positive dependence and a negative one induces negative dependence.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

combine_networks *A single block-diagonal network created by combining multiple networks*

Description

Given a list of compatible networks, the `combine_networks()` returns a single block-diagonal network, preserving attributes that can be preserved.

Usage

```
combine_networks(
  nwl,
  ignore.nattr = c("mnext"),
  ignore.vattr = c(),
  ignore.eattr = c(),
  blockID.vattr = ".NetworkID",
  blockName.vattr = NULL,
  detect.edgecov = FALSE,
  keep.unshared.attr = FALSE,
  subnet.cache = FALSE
)

## S3 method for class 'combined_networks'
print(x, ...)

## S3 method for class 'combined_networks'
summary(object, ...)

## S3 method for class 'summary.combined_networks'
print(x, ...)
```

Arguments

nwl	a list of network::networks to be combined; they must have similar fundamental properties: directedness and bipartedness, though their sizes (and the size of each bipartite group) can vary.
ignore.nattr, ignore.vattr, ignore.eattr	network, vertex, and edge attributes not to be processed as described below.
blockID.vattr	name of the vertex attribute into which to store the index of the network to which that vertex originally belonged.
blockName.vattr	if not NULL, the name of the vertex attribute into which to store the name of the network to which that vertex originally belonged.
detect.edgecov	if TRUE, combine network attributes that look like dyadic covariate (ergm::edgecov) matrices into a block-diagonal matrix.
keep.unshared.attr	whether to keep those network, vertex, and edge attributes not shared by all networks in the list; if TRUE, positions corresponding to networks lacking the attribute are replaced with NA, NULL, or some other placeholder; incompatible with detect.edgecov==TRUE.
subnet.cache	whether to save the input network list as an attribute of the combined network, so that if the network is resplit using on the same attribute (e.g. using uncombine_network()), an expensive call to split.network() can be avoided, at the cost of storage.
x, object	a combined network.
...	additional arguments to methods.

Value

an object of class `combined_networks` inheriting from `network::network` with a block-diagonal structure (or its bipartite equivalent) comprising the networks passed in `nwl`. In particular,

- the returned network's size is the sum of the input networks';
- its basic properties (directedness and bipartednes) are the same;
- the input networks' sociomatrices (both edge presence and edge attributes) are the blocks in the sociomatrix of the returned network;
- vertex attributes are concatenated;
- edge attributes are assigned to their respective edges in the returned network;
- network attributes are stored in a list; but if `detect.edgecov==TRUE`, those network attributes that have the same dimension as the sociomatrices of the constituent networks, they are combined into a single block-diagonal matrix that is then stored as that attribute.

In addition, two new vertex attributes, specified by `blockID.vattr` and (optionally) `blockName.vattr` contain, respectively, the index in `nwl` of the network from which that vertex came and its name, determined as follows:

1. If `nwl` is a named list, names from the list are used.
2. If not 1, but the network has an attribute `title`, it is used.
3. Otherwise, a numerical index is used.

If `blockID.vattr` already exists on the constituent networks, the index is *prepended* to the attribute.

The values of `blockID.vattr` and `blockName.vattr` are stored in network attributes `".blockID.vattr"` and `".blockName.vattr"`.

Functions

- `print(combined_networks)`: A wrapper around `network::print.network()` to print constituent network information and omit some internal variables.
- `summary(combined_networks)`: A wrapper around `network::summary.network()` to print constituent network information and omit some internal variables.
- `print(summary.combined_networks)`: A wrapper around `network::print.summary.network()` to print constituent network information and omit some internal variables.

Examples

```
data(samplk)

o1 <- combine_networks(list(samplk1, samplk2, samplk3))
image(as.matrix(o1))
head(get.vertex.attribute(o1, ".NetworkID"))
o2 <- combine_networks(list(o1, o1))
image(as.matrix(o2))
head(get.vertex.attribute(o2, ".NetworkID", unlist=FALSE))

data(florentine)
f1 <- combine_networks(list(business=flobusiness, marriage=flomarriage),
```

```
                        blockName.vattr=".NetworkName")
image(as.matrix(f1))
head(get.vertex.attribute(f1, ".NetworkID"))
head(get.vertex.attribute(f1, ".NetworkName"))
```

`control.gofN.ergm` *Auxiliary for Controlling Multinetwork ERGM Linear Goodness-of-Fit Evaluation*

Description

`control.gofN.ergm` (or its alias, `control.gofN`) is intended to be used with `gofN()` specifically and will "inherit" as many control parameters from `ergm` fit as possible().

Usage

```
control.gofN.ergm(
  nsim = 100,
  obs.twostage = nsim/2,
  array.max = 128,
  simulate = control.simulate.ergm(),
  obs.simulate = control.simulate.ergm(),
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)

control.gofN(
  nsim = 100,
  obs.twostage = nsim/2,
  array.max = 128,
  simulate = control.simulate.ergm(),
  obs.simulate = control.simulate.ergm(),
  parallel = 0,
  parallel.type = NULL,
  parallel.version.check = TRUE,
  parallel.inherit.MT = FALSE
)
```

Arguments

`nsim`	Number of networks to be randomly drawn using Markov chain Monte Carlo. This sample of networks provides the basis for comparing the model to the observed network.

`obs.twostage` Either `FALSE` or an integer. This parameter only has an effect if the network has missing data or observational process. For such networks, evaluating the Pearson residual requires simulating the expected value of the conditional variance under the observation process. If `FALSE`, the simulation is performed conditional on the observed network. However, a more accurate estimate can be obtained via a two-stage process:

1. Sample networks from the model without the observational constraint.
2. Conditional on each of those networks, sample with the observational constraint, estimating the variance within each sample and then averaging over the first-stage sample.

Then, `obs.twostage` specifies the number of unconstrained networks to simulate from, which should divide the `control.gofN.ergm()`'s `nsim` argument evenly.

`array.max` Try to avoid creating arrays larger in size (in megabytes) than this. Is ignored if `save_stats` is passed.

`simulate`, `obs.simulate` Control lists produced by `control.simulate.ergm()` or equivalent for unconstrained and constrained simulation, respectively. Parameters are inherited from the model fit and can be overridden here.

`parallel` Number of threads in which to run the sampling. Defaults to 0 (no parallelism). See `ergm-parallel` for details and troubleshooting.

`parallel.type` API to use for parallel processing. Defaults to using the **parallel** package with PSOCK clusters. See `ergm-parallel`.

`parallel.version.check` Logical: If TRUE, check that the version of **ergm** running on the slave nodes is the same as that running on the master node.

`parallel.inherit.MT` Logical: If TRUE, slave nodes and processes inherit the `set.MT_terms()` setting.

Details

Auxiliary function as user interface for fine-tuning ERGM Goodness-of-Fit Evaluation.

`ddspL-ergmTerm` *Dyadwise shared partners on layers*

Description

This term adds one network statistic to the model for each element in `d` where the i th such statistic equals the number of dyads in the network with exactly `d[i]` shared partners. For a directed network, multiple shared partner definitions are possible.

`dspL` and `ddspL` are aliases for consistency with **ergm**.

Usage

```
# binary: ddspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)

# binary: dspL(d, type="OTP", Ls.path=NULL, L.in_order=FALSE)
```

Arguments

`Ls.path`, `L.in_order`
: a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- Outgoing Two-path (`"OTP"`): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path (`"ITP"`): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"
- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

`despL-ergmTerm` *Edgewise shared partners on layers*

Description

This term adds one network statistic to the model for each element in `d` where the i th such statistic equals the number of edges in the network with exactly `d[i]` shared partners. For a directed network, multiple shared partner definitions are possible.

`espL` and `despL` are aliases for consistency with **ergm**.

Usage

```
# binary: despL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)

# binary: espL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)
```

Arguments

`d`	a vector of distinct integers
`type`	A string indicating the type of shared partner or path to be considered for directed networks: `"OTP"` (default for directed), `"ITP"`, `"RTP"`, `"OSP"`, and `"ISP"`; has no effect for undirected. See the section below on Shared partner types for details.
`L.base`	the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specification for the base
`Ls.path, L.in_order`	a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- Outgoing Two-path (`"OTP"`): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path (`"ITP"`): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"

- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \rightarrow k, j \rightarrow k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \rightarrow i, k \rightarrow j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

`dgwdspL-ergmTerm` *Geometrically weighted dyadwise shared partner distribution on layers*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution with decay parameter. Note that the GWDSP statistic is equal to the sum of GWNSP plus GWESP. For a directed network, multiple shared partner definitions are possible.

`gdwdspL` and `dgwdspL` are aliases for consistency with **ergm**.

Usage

```
# binary: dgwdspL(decay, fixed=FALSE, cutoff=30, type="OTP",
#                 Ls.path=NULL, L.in_order=FALSE)

# binary: gwdspL(decay, fixed=FALSE, cutoff=30, type="OTP",
#                Ls.path=NULL, L.in_order=FALSE)
```

Arguments

`decay`	nonnegative decay parameter for the shared partner or selected directed analogue count; required if `fixed=TRUE` and ignored with a warning otherwise.
`fixed`	optional argument indicating whether the `decay` parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE` , which means the scale parameter is not fixed and thus the model is a curved exponential family.
`cutoff`	This optional argument sets the number of underlying DSP terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See `?control.ergm`.)
`type`	A string indicating the type of shared partner or path to be considered for directed networks: `"OTP"` (default for directed), `"ITP"`, `"RTP"`, `"OSP"`, and `"ISP"`; has no effect for undirected. See the section below on Shared partner types for details.
`Ls.path, L.in_order`	a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- Outgoing Two-path (`"OTP"`): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path (`"ITP"`): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"
- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

---

dgwespL-ergmTerm *Geometrically weighted edgewise shared partner distribution on layers*

Description

This term adds a statistic equal to the geometrically weighted edgewise (not dyadwise) shared partner distribution with decay parameter. For a directed network, multiple shared partner definitions are possible.

gdwespL and dgwespL are aliases for consistency with **ergm**.

Usage

```
# binary: dgwespL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#                 Ls.path=NULL, L.in_order=FALSE)

# binary: gwespL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#                Ls.path=NULL, L.in_order=FALSE)
```

Arguments

decay	nonnegative decay parameter for the shared partner or selected directed analogue count; required if fixed=TRUE and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is FALSE , which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying ESP terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)
type	A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.

`L.base` the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specification for the base

`Ls.path`, `L.in_order`
a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- Outgoing Two-path (`"OTP"`): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path (`"ITP"`): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"
- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

dgwnspL-ergmTerm *Geometrically weighted non-edgewise shared partner distribution on layers*

Description

This term is just like gwespL and gwdspL except it adds a statistic equal to the geometrically weighted nonedgewise (that is, over dyads that do not have an edge) shared partner distribution with decay parameter. For a directed network, multiple shared partner definitions are possible.

gdwnspL and dgwnspL are aliases for consistency with **ergm**.

Usage

```
# binary: dgwnspL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#                 Ls.path=NULL, L.in_order=FALSE)

# binary: gwnspL(decay, fixed=FALSE, cutoff=30, type="OTP", L.base=NULL,
#                Ls.path=NULL, L.in_order=FALSE)
```

Arguments

decay	nonnegative decay parameter for the shared partner or selected directed analogue count; required if fixed=TRUE and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is FALSE , which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying NSP terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)
type	A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.
L.base	the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specification for the base
Ls.path, L.in_order	a vector of one or two formulas Ls.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order==TRUE , the first tie of the two-path must be the first element of Ls.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP" , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the `type` argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the `relevent` package):

- Outgoing Two-path (`"OTP"`): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path (`"ITP"`): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"
- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \to k, j \to k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \to i, k \to j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

`direct.network` *Returns a directed version of an undirected binary network*

Description

Returns a directed version of an undirected binary network

Usage

```
direct.network(x, rule = c("both", "upper", "lower"))
```

Arguments

`x`	a `network` object.
`rule`	a string specifying how the network is to be constructed.

dnspL-ergmTerm *Non-edgewise shared partners and paths on layers*

Description

This term adds one network statistic to the model for each element in d where the i th such statistic equals the number of non-edges in the network with exactly d[i] shared partners. For a directed network, multiple shared partner definitions are possible.

nspL and dnspL are aliases for consistency with **ergm**.

Usage

```
# binary: dnspL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)

# binary: nspL(d, type="OTP", L.base=NULL, Ls.path=NULL, L.in_order=FALSE)
```

Arguments

d	a vector of distinct integers
type	A string indicating the type of shared partner or path to be considered for directed networks: "OTP" (default for directed), "ITP", "RTP", "OSP", and "ISP"; has no effect for undirected. See the section below on Shared partner types for details.
L.base	the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specification for the base
Ls.path, L.in_order	a vector of one or two formulas Ls.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order==TRUE , the first tie of the two-path must be the first element of Ls.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP" , the first tie is considered to be the one incident on the tail of the base tie.)

Shared partner types

While there is only one shared partner configuration in the undirected case, nine distinct configurations are possible for directed graphs, selected using the type argument. Currently, terms may be defined with respect to five of these configurations; they are defined here as follows (using terminology from Butts (2008) and the relevent package):

- Outgoing Two-path ("OTP"): vertex k is an OTP shared partner of ordered pair (i, j) iff $i \to k \to j$. Also known as "transitive shared partner".
- Incoming Two-path ("ITP"): vertex k is an ITP shared partner of ordered pair (i, j) iff $j \to k \to i$. Also known as "cyclical shared partner"

- Reciprocated Two-path (`"RTP"`): vertex k is an RTP shared partner of ordered pair (i, j) iff $i \leftrightarrow k \leftrightarrow j$.
- Outgoing Shared Partner (`"OSP"`): vertex k is an OSP shared partner of ordered pair (i, j) iff $i \rightarrow k, j \rightarrow k$.
- Incoming Shared Partner (`"ISP"`): vertex k is an ISP shared partner of ordered pair (i, j) iff $k \rightarrow i, k \rightarrow j$.

By default, outgoing two-paths (`"OTP"`) are calculated. Note that Robins et al. (2009) define closely related statistics to several of the above, using slightly different terminology.

Note

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

`Goeyvaerts` *A sample of within-household contact networks in Flanders and Brussels*

Description

This is a list of 318 `network` objects derived from contact diary data collected by by Goeyvaerts et al. (2018). The study recruited households in Flanders and Brussels-Capital region with at least one child 12 or under. The networks are symmetrized.

Usage

```
data(Goeyvaerts)
```

Format

An object of class `list` of length 318.

Nonstandard Network Attributes

`included` (logical) whether the network was included in Goeyvaerts's analysis. (Two were excluded.)

`weekday` (logical) whether the contact diary on which the network is based was collected on a weekday, as opposed to weekend.

Nonstandard Vertex Attributes

`age` (numeric) the household member's age.

`gender` (character) the household member's gender (`"F"`/`"M"`).

`role` (character) the household member's inferred role (`"Father"`/`"Mother"`/`"Child"`/`"Grandmother"`).

Licenses and Citation

When publishing results obtained using this data set, the original authors (Goeyvaerts et al. 2018) should be cited, along with this R package.

Source

The data were collected and by Goeyvaerts et al. (2018) and curated by Pietro Coletti.

References

Goeyvaerts N, Santermans E, Potter G, Torneri A, Kerckhove KV, Willem L, Aerts M, Beutels P, Hens N (2018). "Household Members Do Not Contact Each Other at Random: Implications for Infectious Disease Modelling." *Proceedings of the Royal Society B: Biological Sciences*, **285**(1893), 20182201. doi:10.1098/rspb.2018.2201.

See Also

`vignette("Goeyvaerts_reproduction")` for a vignette reproducing the Goeyvaerts analysis and performing diagnostics

---

gofN *Linear model diagnostics for multinetwork linear models*

Description

`gofN()` performs a simulation to obtain Pearson residuals for the multivariate linear model for ERGM parameters, which can then be used for a variety of diagnostics and diagnostic plots developed by Krivitsky et al. (2023).

Usage

```
gofN(
  object,
  GOF = NULL,
  subset = TRUE,
  control = control.gofN.ergm(),
  save_stats = FALSE,
  ...
)

## S3 method for class 'gofN'
```

```
x[i, j, ..., drop = FALSE]

## S3 method for class 'gofN'
augment(x, ...)

## S3 method for class 'gofN'
summary(object, by = NULL, ...)
```

Arguments

object	an ergm object.
GOF	a one-sided ergm formula specifying network statistics whose goodness of fit to test, or NULL; if NULL, uses the original model.
subset	argument for the N term.
control	See control.gofN.ergm().
save_stats	If TRUE, save the simulated network statistics; defaults to FALSE to save memory and disk space.
...	additional arguments to functions (simulate.ergm() and summary.ergm_model()) for the constructor.
x	a gofN object.
i	for the indexing operator, index of statistics to be kept in the subset.
j	for the indexing operator, index of networks to be kept in the subset.
drop	whether the indexing operator should drop attributes and return simply a list.
by	a numeric or character vector, or a formula whose RHS gives an expression in terms of network attributes, used as a grouping variable for summarizing the values.

Value

An object of class gofN: a named list containing a list for every statistic in the specified GOF formula with the following elements vectors of length equal to the number of subnetworks:

observed	For completely observed networks, their value of the statistic. For partially observed networks, the expected value of their imputations under the model.
fitted	Expected value of the statistic under the model.
var	Variance of the statistic under the model.
var.obs	Conditional variance under imputation statistic.
pearson	The Pearson residual computed from the above.
stats, stats.obs	If save_stats control parameter is TRUE, the simulated statistics.

In addition, the following attr-style attributes are included:

nw	The observed multinetwork object.
subset	A logical vector giving the subset of networks that were used.
control	Control parameters passed.

Methods (by generic)

- `[`: Extract a subset of statistics for which goodness-of-fit had been computed.
- `augment(gofN)`: a method for constructing a `tibble` of network attributes augmented with goodness of fit information. Columns include:

  network attributes the attributes of each of the networks

  `.stat_name` name of the simulated statistic

  `.stat_id` index of the simulated statistic in the `gofN` object

  `.network_id` index of the network in the networks for which `gofN` was run (excluding those not in the subset)

  `.fitted` predicted value for the statistic

  `.observed` either the observed (for completely observed networks) or the predicted conditional on observed (for partially observed networks) value of the statistic

  `.pearson` the standardised Pearson residual

  `.var`, `.var.obs` estimated unconditional and average conditional variance of the statistic

  `.weight` inverse of the variance of the residual
- `summary(gofN)`: A simple summary function.

References

Krivitsky PN, Coletti P, Hens N (2023). "A Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks." *Journal of the American Statistical Association*, **118**(544), 2213-2224. doi:10.1080/01621459.2023.2242627.

See Also

`plot.gofN()` and `autoplot.gofN()` for plotting `gofN` objects to make residual plots; `ergm::gof()` for single-network goodness-of-fit simulations in **ergm**

Examples

```
data(samplk)
monks <- Networks(samplk1, samplk2, samplk3,samplk1, samplk2, samplk3,samplk1, samplk2, samplk3)
fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model
summary(fit.gof)
plot(fit.gof)
fit.gof <- gofN(fit, GOF=~triangles)
summary(fit.gof)
plot(fit.gof)


samplk1[1,]<-NA
samplk2[,2]<-NA
monks <- Networks(samplk1, samplk2, samplk3,samplk1, samplk2, samplk3,samplk1, samplk2, samplk3)
fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model
summary(fit.gof)
plot(fit.gof)
```

```
fit.gof <- gofN(fit, GOF=~triangles)
summary(fit.gof)
plot(fit.gof)
plot(fit.gof, against=~log(.fitted)) # Plot against transformed fitted values.


### If 'ggplot2' and 'ggrepel' are installed, illustrate the autoplot() method.
if(require("ggplot2") && requireNamespace("ggrepel")){
  autoplot(fit.gof)
}

# Default is good enough in this case, but sometimes, we might want to set it higher. E.g.,
## Not run:
fit.gof <- gofN(fit, GOF=~edges, control=control.gofN.ergm(nsim=400))

## End(Not run)


### If 'generics' is installed, illustrate the augment() method.
if(require("generics")){
  augment(fit.gof)
}
```

`gwb1dspL-ergmTerm` *Geometrically weighted dyadwise shared partner distribution for dyads in the first bipartition on layers*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution for dyads in the first bipartition with decay parameter `decay` parameter, which should be non-negative. This term can only be used with bipartite networks.

Usage

```
# binary: gwb1dspL(decay=0, fixed=FALSE, cutoff=30, Ls.path=NULL)
```

Arguments

`decay`	nonnegative decay parameter for the shared partner counts; required if `fixed=TRUE` and ignored with a warning otherwise.
`fixed`	optional argument indicating whether the `decay` parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is `FALSE` , which means the scale parameter is not fixed and thus the model is a curved exponential family.

`cutoff` This optional argument sets the number of underlying `b1dsp` terms to use in computing the statistics when `fixed=FALSE`, in order to reduce the computational burden. Its default value can also be controlled by the `gw.cutoff` term option control parameter. (See `?control.ergm`.)

`Ls.path`, `L.in_order`
a vector of one or two formulas `Ls.path` provides the Layer Logic (c.f. Layer Logic section in the `Layer()` documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If `L.in_order==TRUE` , the first tie of the two-path must be the first element of `Ls.path` and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types `"OSP"` and `"ISP"` , the first tie is considered to be the one incident on the tail of the base tie.)

Note

This term does not support multilayer networks with heterogeneous bipartedness. This may change in the future.

This term takes an additional term option (see `options?ergm`), `cache.sp`, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: bipartite, curved, layer-aware, undirected, binary

`gwb2dspL-ergmTerm` *Geometrically weighted dyadwise shared partner distribution for dyads in the second bipartition on layers*

Description

This term adds one network statistic to the model equal to the geometrically weighted dyadwise shared partner distribution for dyads in the second bipartition with decay parameter `decay` parameter, which should be non-negative. This term can only be used with bipartite networks.

Usage

```
# binary: gwb2dspL(decay=0, fixed=FALSE, cutoff=30, Ls.path=NULL)
```

Arguments

decay	nonnegative decay parameter for the shared partner counts; required if fixed=TRUE and ignored with a warning otherwise.
fixed	optional argument indicating whether the decay parameter is fixed at the given value, or is to be fit as a curved exponential-family model (see Hunter and Handcock, 2006). The default is FALSE , which means the scale parameter is not fixed and thus the model is a curved exponential family.
cutoff	This optional argument sets the number of underlying b2dsp terms to use in computing the statistics when fixed=FALSE, in order to reduce the computational burden. Its default value can also be controlled by the gw.cutoff term option control parameter. (See ?control.ergm.)
Ls.path, L.in_order	a vector of one or two formulas Ls.path provides the Layer Logic (c.f. Layer Logic section in the Layer() documentation) specifications for the ties of the 2-path or the shared partnership. (If only one formula is given the layers are assumed to be the same.) If L.in_order==TRUE , the first tie of the two-path must be the first element of Ls.path and the second must be the second; otherwise, any ordering counts, provided there is exactly one of each. (For types "OSP" and "ISP" , the first tie is considered to be the one incident on the tail of the base tie.)

Note

This term does not support multilayer networks with heterogeneous bipartedness. This may change in the future.

This term takes an additional term option (see options?ergm), cache.sp, controlling whether the implementation will cache the number of shared partners for each dyad in the network; this is usually enabled by default.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: bipartite, curved, layer-aware, undirected, binary

L-ergmTerm *Evaluation on layers*

Description

Evaluates the terms in formula on an observed or logical layers specified in formula Ls and sums the results elementwise.

Usage

```
# binary: L(formula, Ls=~.)
```

Arguments

`formula`	a one-sided `ergm()`-style formula with the terms to be evaluated
`Ls`	either a Layer Logic specification formula (c.f. Layer Logic section in the `Layer()` documentation) or a list thereof (constructed by `list()` or `c()`), on which to evaluate `formula`

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: layer-aware, operator, binary

`Layer` *A multilayer network representation.*

Description

A function for specifying the LHS of a multilayer (a.k.a. multiplex, a.k.a. multirelational, a.k.a. multivariate) ERGM in the framework of Krivitsky et al. (2020).

Usage

```
Layer(..., .symmetric = NULL, .bipartite = NULL, .active = NULL)
```

Arguments

`...`	layer specification, in one of three formats: 1. An (optionally named) list of identically-dimensioned networks. 2. Several networks as (optionally named) arguments. 3. A single network, a character vector, and several optional arguments. Then, the layers are values of the named edge attributes. If the vector has named elements (e.g., `c(a="advice", c="collaboration")`), the layers will be renamed accordingly. The optional arguments `.symmetric` and `.bipartite` are then interpreted as described below.
`.symmetric`	If the layer specification is via a single network with edge attributes and the network is directed, an optional logical vector to specify which of the layers should be treated as undirected.
`.bipartite`	If the layer specification is via a single network with edge attributes and the network is unipartite, an optional integer vector to specify which of the layers should be treated as bipartite and how many mode 1 vertices there are.
`.active`	An optional list with a nodal attribute specification (`? nodal_attributes`) for each layer, specifying which nodes on each layer *may* have ties.

Value

A network object comprising the provided layers, with layer metadata.

Due to certain optimizations, the resulting network object's network and vertex attributes should be treated as read-only: do not modify them. If you need to change existing attributes or add new ones, do so on the input networks and call `Layer(...)` again.

Specifying models for multilayer networks

In order to fit a model for multilayer networks, first use Layer construct an LHS network that ergm() will understand as multilayered.

Used in the formula directly, most, but not all, **ergm** terms will sum their statistics over the observed layers.

Some terms are *layer-aware*, however. 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 c()) is named `Ls=`; and a specification of the form `~.` is a placeholder for all observed layers.

Operator `L(formula, Ls=...)` can be used to evaluate arbitrary terms in the `formula` on specified layers.

Layer specification documentation follows.

Layer Logic:

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` ``).

Arithmetical, relational, and logical operators can be used to combine them. All listed operators are implemented, as well as functions abs, round, and sign. Standard operator precedence applies, so use of parentheses is recommended to ensure the logical expression is what it looks like.

Important: For performance reasons, **ergm.multi**'s Layer Logic implementation uses integer arithmetic. This means, in particular, that `/` will round down instead of returning a fraction (as `%/%` does in R), and round() function without a second argument (which can be negative to round to the nearest 10, 100, etc.) is not meaningful and will be ignored.

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(`1`)&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.

Summing layers:

Some of the terms that call for a list of layers (i.e., have `Ls=` arguments) will sum the statistic over the layers. For example, `Layer(nw1,nw2)~L(~edges, c(~`1`,~(`2`&!`1`)))` produces the number of edges in layer 1 plus the number of edges in layer 2 but not in layer 1.

For these formulas, one can specify the layer's weight on its left-handside. For example, `Layer(nw1,nw2)~L(~edges, c(3~`1`,-1~(`2`&!`1`)))` will produce three times the number of edges in layer 1, minus the number of edges in layer 2 but not in layer 1.

Note

The resulting network will be the "least common denominator" network: if not all layers have the same bipartedness, all layers will appear as unipartite to the statistics, and if any are directed, all will be. However, certain operator terms, particularly `Symmetrize()` and `S()`, can be used to construct a bipartite subgraph of a unipartite graph or change directedness.

Its nonstandard network and vertex attributes will be taken from the *first* network in the list. The subsequent networks' attributes will be overwritten with a warning if they differ from those in the first network.

References

Krivitsky PN, Koehly LM, Marcum CS (2020). "Exponential-family Random Graph Models for Multi-layer Networks." *Psychometrika*, **85**(3), 630–659. doi:10.1007/s11336020097207.

See Also

Help on model specification for specific terms.

Examples

```
data(florentine)

# Method 1: list of networks
flo <- Layer(list(m = flomarriage, b = flobusiness))
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

# Method 2: networks as arguments
flo <- Layer(m = flomarriage, b = flobusiness)
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

# Method 3: edge attributes (also illustrating renaming):
flo <- flomarriage | flobusiness
flo[,, names.eval="marriage"] <- as.matrix(flomarriage)
flo[,, names.eval="business"] <- as.matrix(flobusiness)
flo # edge attributes
flo <- Layer(flo, c(m="marriage", b="business"))
ergm(flo ~ L(~edges, ~m)+L(~edges, ~b))

### Specifying modes and mixed bipartitedness

# Suppose we have a two-mode network with 5 nodes on Mode 1 and 15
# on Mode 2, and suppose that we observe two layers, one only among
# actors of Mode 1 and the other bipartite between Modes 1 and 2.
```

```
# Construct the two layers' networks:
nw1 <- network.initialize(20, dir=FALSE)
nw12 <- network.initialize(20, dir=FALSE, bipartite=5)
nw1 %v% "mode" <- rep(1:2,c(5,15))

# For testing: the maximal set of edges for each type of network:
nw1[1:5,1:5] <- 1
nw12[1:5,6:20] <- 1

# The .active argument specifies the following:
# * nw1's vertices are only active if their mode=1 (i.e., 1-2, 2-1,
#   and 2-2 can't have edges).
# * nw12's vertices are all active, but the network is bipartite,
#   so constraints will be adjusted automatically.
lnw <- Layer(nw1, nw12, .active=list(~mode==1, ~TRUE))

summary(lnw~
edges+ # 5*4/2+5*15 = 10+75 = 85
L(~edges,~`1`)+ # 5*4/2 = 10
L(~edges,~`2`)+ # 5*15 = 75
L(~edges,~(`1`|`2`))+ # This logical layer has contents of both, so also 85.
L(~edges,~(`1`&`2`)) # There is no overlap between the two layers, so 0.
)

# Layer-aware terms can be used:

nw1[,] <-0
nw1[1,2:3] <- 1
nw1[2,3] <- 1
nw12[,] <- 0
nw12[1,6:7] <- 1
nw12[2,6:7] <- 1

lnw <- Layer(nw1, nw12, .active=list(~mode==1,~TRUE))

summary(lnw~L(~triangles, ~`1`)+ # 1-2-3 triangle.
  L(~triangles, ~`1`|`2`)+ # 1-2-3, 1-2-6, 1-2-7 triangles
  dgwespL(L.base=~`1`, Ls.path=list(~`2`,~`2`)) # 1-2-6 and 1-2-7 only
)

# Because the layers are represented as a block-diagonal matrix,
# this will only count triangles entirely contained within a single
# layer, i.e., 1-2-3:
summary(lnw~triangles)

# If you need to evaluate bipartite-only statistics on the second
# layer, you need to use the S() operator to select the bipartite
# view:
summary(lnw~L(~S(~b1degree(1:3)+b2degree(1:3),1:5~6:20), ~`2`))
```

Lazega *A network of advice, collaboration, and friendship in a law firm*

Description

This dataset contains a network of relations of various types among 71 lawyers (partners and associates) in a New England (Northeastern US) corporate law firm referred to as "SG&R" collected 1988–1991 by Lazega (2001).

Usage

```
data(Lazega)
```

Format

An object of class network of length 5.

Details

All relations are directed.

Nonstandard Vertex Attributes

age (numeric) the lawyer's age.

gender (character) the lawyer's gender ("man"/"woman").

office (character) in which of the firm's three offices the lawyer is based ("Boston"/"Hartford"/"Providence").

practice (character) which area of law the lawyer practices ("corporate"/"litigation").

school (character) from which law school the lawyer graduated ("Harvard/Yale"/"UConnecticut"/"other").

seniority (numeric) the lawyer's seniority rank in the firm (1 = high).

status (character) the lawyer's status in the firm ("associate"/"partner").

yrs_frm (numeric) the number of years the lawyer has been with the firm.

Nonstandard Edge Attributes

Each directed edge $i \to j$ has the following attributes:

advice (logical) whether i has reported receiving advice from j. (Note that as defined, advice flows from head of the directed edge to the tail.)

coworker (logical) whether i has reported receiving j's assistance in preparing documents. (Note that as defined, assistance flows from head of the directed edge to the tail.)

friendship (logical) whether i considers j a friend outside of work.

Licenses and Citation

When publishing results obtained using this data set, the original author (Lazega 2001) should be cited, along with this R package.

Source

This version of the dataset was retrieved from the RSiena web site and was compiled by Christopher Steven Marcum and Pavel N. Krivitsky for Krivitsky et al. (2020).

References

Krivitsky PN, Koehly LM, Marcum CS (2020). "Exponential-family Random Graph Models for Multi-layer Networks." *Psychometrika*, **85**(3), 630–659. doi:10.1007/s11336020097207.

Lazega E (2001). *The Collegial Phenomenon: The Social Mechanisms of Cooperation among Peers in a Corporate Law Partnership*. Oxford University Press. ISBN 9780199242726.

Examples

```
data(Lazega)
# Construct a multilayer network for ergm(). (See `?Layer` for syntax.)
LLazega <- Layer(Lazega, c("advice", "coworker", "friendship"))
# Specify a layer logic model.
efit <- ergm(LLazega ~ L(~edges + mutual, ~advice) +
                       L(~edges + mutual, ~coworker) +
                       L(~edges + mutual, ~friendship) +
                       L(~edges + mutual, ~advice&coworker) +
                       L(~edges + mutual, ~advice&friendship) +
                       L(~edges + mutual, ~coworker&friendship))
summary(efit)
```

`lm.gofN` *Fit a linear model to the residuals in a gofN object.*

Description

This non-method runs a properly weighted linear model on the raw residuals of a `gofN` simulation for a multi-network ERGM fit.

Usage

```
lm.gofN(formula, data, ...)
```

Arguments

`formula`	an `lm`-style formula. See Details for interpretation.
`data`	a `gofN` object.
`...`	additional arguments to `lm()`, excluding `weights`.

Details

The formula's RHS is evaluated in an environment comprising the network statistics used in the gofN() call (which refer to the raw residuals for the corresponding statistic) and the network attributes.

The LHS is handled in a nonstandard manner, designed to make it easier to reference the usually lengthy network statistics: first, it is evaluated in the formula's environment. If the evaluation is successful and the result is numeric, these numbers are used as indices of the statistics in the gofN object to use on the RHS. If it is a character vector, it is treated as names of these statistics.

Value

A list of lm objects, one for each element of the vector on the LHS.

See Also

gofN() and related methods.

Examples

```
data(samplk)
# Add time indices:
samplk1 %n% "t" <- 1
samplk2 %n% "t" <- 2
samplk3 %n% "t" <- 3

monks <- Networks(samplk1, samplk2, samplk3)

fit <- ergm(monks~N(~edges+nodematch("group")))
fit.gof <- gofN(fit) # GOF = original model

# Is there a time effect we should incorporate?
fit.gof.lm <- lm.gofN((1:2)~t, data=fit.gof)

lapply(fit.gof.lm, summary)
```

---

marg_cond_sim *Calculate gofN()-style Pearson residuals for arbitrary statistics*

Description

This function is to be considered experimental. Do NOT rely on it. It may, eventually, be moved to ergm, perhaps integrated into the simulate methods.

Usage

```
marg_cond_sim(
  object,
  nsim = 1,
  obs.twostage = nsim/2,
  GOF = NULL,
  control = control.gofN.ergm(),
  save_stats = FALSE,
  negative_info = c("error", "warning", "message", "ignore"),
  ...
)
```

Arguments

object	an ergm object.
nsim	number of realizations.
obs.twostage, GOF, save_stats	see gofN().
control	a control list returned by control.gofN.ergm(); note that nsim and obs.twostage parameters in the control list are ignored in favor of those passed to the function directly.
negative_info	how to handle the situation in which the constrained variance exceeds the unconstrained: the corresponding action will be taken.
...	additional arguments to ergm_model(), simulate.ergm(), and summary.ergm_model().

Value

an object of similar structure as that returned by gofN().

---

mutualL-ergmTerm *Mutuality*

Description

In binary ERGMs, equal to the number of pairs of actors i and j for which $(i \to j)$ and $(j \to i)$ both exist.

Usage

```
# binary: mutualL(same=NULL, diff=FALSE, by=NULL, keep=NULL, Ls=NULL)
```

Arguments

same	optional argument. If passed the name of a vertex attribute, only mutual pairs that match on the attribute are counted. Only one of same or by may be used. If both parameters are passed, same takes precedent. This parameter is affected by diff.
diff	separate counts for each unique matching value can be obtained by using diff=TRUE with same.
by	each node is counted separately for each mutual pair in which it occurs and the counts are tabulated by unique values of the attribute if passed the name of a vertex attribute. This means that the sum of the mutual statistics when by is used will equal twice the standard mutual statistic. Only one of same or by may be used. If both parameters are passed, same takes precedent. This parameter is not affected by diff.
keep	a numerical vector to specify which statistics should be kept whenever the mutual term would ordinarily result in multiple statistics.
Ls	a list (constructed by list() or c() of one or two Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation). If given, the statistic will count the number of dyads where a tie in Ls[[1]] reciprocates a tie in Ls[[2]] and vice versa. (Note that dyad that has mutual ties in Ls[[1]] and in Ls[[2]] will add 2 to this statistic.) If a formula is given, it is replicated.

Details

This term can only be used with directed networks.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: directed, frequently-used, layer-aware, binary

N-ergmTerm *Evaluation on multiple networks*

Description

Evaluates the terms in formula on each of the networks joined using Networks function, and returns either a weighted sum or an lm-style linear model for the ERGM coefficients (Krivitsky et al. 2023). Its syntax follows that of lm closely, with sensible defaults.

The default formula (~1) sums the specified network statistics. If lm refers to any network attributes for which some networks have missing values, the term will stop with an error. This can be avoided by pre-filtering with subset, which controls which networks are affected by the term.

The formula may also reference .NetworkID and .NetworkName. In particular, ~0+factor(.NetworkID) will evaluate formula on each network individually.

Usage

```
# binary: N(formula, lm=~1, subset=TRUE, weights=1, contrasts=NULL, offset=0, label=NULL,
#           .NetworkID=".NetworkID", .NetworkName=".NetworkName")

# valued: N(formula, lm=~1, subset=TRUE, weights=1, contrasts=NULL, offset=0, label=NULL,
#           .NetworkID=".NetworkID", .NetworkName=".NetworkName")
```

Arguments

`.NetworkID`, `.NetworkName`	Optional strings indicating the vertex attributes used to distinguish and name the networks; intended to be used by term developers.
`formula`	a one-sided `ergm()`-style formula with the terms to be evaluated
`lm`	a one-sided `lm()`-style formula whose RHS specifies the network-level predictors for the terms in the `ergm()` formula `formula`.
`subset`, `contrasts`	see `lm()`.
`offset`	A constant, a vector of length equal to the number of networks, or a matrix whose number of rows is the number of networks and whose number of columns is the number of free parameters of the ERGM. It can be specified in `lm` as well.
`weights`	reserved for future use; attempting to change it will cause an error: at this time, there is no way to assign sampling weights to networks.
`label`	An optional parameter which will add a label to model parameters to help identify the term (which may have similar predictors but, say, a different network subset) in the output *or* a function that wraps the names.

Offsets and fixing parameters

By default, an `N(formula, lm)` term will add $p \times q$ free parameters, where p is the number of free parameters (possibly curved) of the ERGM specified by `formula`, and q is the number of parameters specified by the `lm` formula. That is, there would be one parameter for each combination of an ERGM parameter and a linear model parameter, in an ERGM-major order (i.e., for each ERGM parameter, the linear model parameters will be enumerated). For example, the term `gwesp()` has two free parameters: its coefficient and its decay rate. We can specify a model in which they depend on $\log(n)$ as `N(~gwesp, ~log(n))`, resulting in the following 4 parameters, with the intercept for the linear model being implicit:

```
#> [1] "N(1)~gwesp"            "N(log(n))~gwesp"       "N(1)~gwesp.decay"
#> [4] "N(log(n))~gwesp.decay"
```

If a different linear model is desired for different ERGM terms (e.g., some are to be affected by network size while others are not), multiple `N()` terms can be specified. This covers most such cases, but not all. For example, suppose that for the above model, we wish for its coefficient to depend on `log(n)` but for the decay parameter not to. In this case, one can use the `offset()` decorator with partial offsetting. Then, specifying `offset(N(~gwesp(), ~log(n)), 4)`, we get:

```
#> [1] "N(1)~gwesp"                    "N(log(n))~gwesp"
#> [3] "N(1)~gwesp.decay"              "offset(N(log(n))~gwesp.decay)"
```

Then, setting the corresponding `offset.coef = 0` will fix the coefficient of `log(n)` for the decay parameter at 0, while allowing a constant decay parameter to be estimated.

Note

Care should be taken to avoid multicollinearity when using this operator. As with the `lm()` function, `lm` formulas have an implicit intercept, which can be suppressed by specifying `~ 0 + ...` or `~ -1 + ...` on the formula. When `lm` is given a model with intercept and a categorical predictor (including a `logical` one), it will use the first level (or `FALSE`) as the baseline, but if the model is without intercept, it will use all levels of the first categorical predictor. This is typically what is wanted in a linear regression, but for the `N` operator, this can be problematic if the "intercept" effect is added by a different term. A workaround is to convert the categorical predictor to dummy variables before putting it into the `lm` formula.

References

Krivitsky PN, Coletti P, Hens N (2023). "A Tale of Two Datasets: Representativeness and Generalisability of Inference for Samples of Networks." *Journal of the American Statistical Association*, **118**(544), 2213-2224. doi:10.1080/01621459.2023.2242627.

See Also

`ergmTerm` for index of model terms currently visible to the package.

Keywords: directed, operator, undirected, binary, valued

`vignette("Goeyvaerts_reproduction")` for a demonstration.

`Networks` *A multinetwork network representation.*

Description

A function for specifying the LHS of a multi-network (a.k.a. multilevel) ERGM. Typically used in conjunction with the `N()` term operator.

Usage

```
Networks(...)
```

Arguments

`...`	network specification, in one of two formats: 1. An (optionally named) list of networks with same directedness and bipartedness (but possibly different sizes). 2. Several networks as (optionally named) arguments.

Value

A network object comprising the provided networks, with multinetwork metadata.

Due to certain optimizations, the resulting network object's network and vertex attributes should be treated as read-only: do not modify them. If you need to change existing attributes or add new ones, do so on the input networks and call `Networks(...)` again.

See Also

ergmTerm for specific terms.

`vignette("Goeyvaerts_reproduction")` for a demonstration.

Examples

```
data(samplk)

# Method 1: list of networks
monks <- Networks(list(samplk1, samplk2))
ergm(monks ~ N(~edges))

# Method 2: networks as arguments
monks <- Networks(samplk1, samplk2)
ergm(monks ~ N(~edges))
```

---

network_view *Construct a "view" of a network.*

Description

Returns a network with edges optionally filtered according to a specified criterion and with edge attributes optionally computed from other edge attributes.

Usage

```
network_view(x, ..., .clear = FALSE, .sep = ".")
```

Arguments

x	a network object.
...	a list of attribute or filtering specifications. See Details.
.clear	whether the edge attributes not set by this call should be deleted.
.sep	when specifying via a character vector, use this as the separator for concatenating edge values.

Details

Attribute specification arguments have the form <newattrname> = <expr>, where <newattrname> specifies the name of the new edge attribute (or attribute to be overwritten) and <expr> can be one of the following:

a function The function will be passed two arguments, the edgelist tibble and the network, and must return a vector of edge attribute values to be set on the edges in the order specified.

a formula The expression on the RHS of the formula will be evaluated with names in it referencing the edge attributes. The input network may be referenced as .nw. The expression's result is expected to be a vector of edge attribute values to be set on the edges in the order specified.

a character vector If of length one, the edge attribute with that name will simply be copied; if greater than one, the attribute values will be concatenated wtih the .sep argument as the separator.

an object enclosed in I() The object will be used directly to set the edge attribute.

Filtering arguments are specified the same way as attribute arguments, but they must be unnamed arguments (i.e., must be passed without the =) and must return a logical or numeric vector suitable for indexing the edge list. Multiple filtering arguments may be specified, and the edge will be kept if it satisfies *all*. If the conjunction of the edge's original states and the filtering results is ambiguous (i.e., NA), it will be set as missing.

Value

A network object with modified edges and edge attributes.

Examples

```
data(florentine)
flo <- flomarriage
flo[,,add.edges=TRUE] <- as.matrix(flomarriage) | as.matrix(flobusiness)
flo[,, names.eval="m"] <- as.matrix(flomarriage)==1
flobusiness[3,5] <- NA
flo[,, names.eval="b"] <- as.matrix(flobusiness)==1
flo
(flob <- network_view(flo, "b"))
(flobusiness) # for comparison


(flob <- network_view(flo, ~b&m))
(flobusiness & flomarriage) # for comparison


as.matrix(flob <- network_view(flo, bm=~b+m), attrname="bm")
(as.matrix(flobusiness) + as.matrix(flomarriage)) # for comparison


as.matrix(flob <- network_view(flo, ~b, bm=~b+m), attrname="bm")
as.matrix(flobusiness)*(1+as.matrix(flomarriage)) # for comparison
```

plot.gofN *Plotting methods for* gofN*, making residual and scale-location plots.*

Description

The plot() method uses R graphics.

The ggplot2::autoplot() method uses **ggplot2** and **ggrepel**.

Usage

```
## S3 method for class 'gofN'
plot(
  x,
  against = NULL,
  which = 1:2,
  col = 1,
  pch = 1,
  cex = 1,
  bg = 0,
  ...,
  ask = length(which) > 1 && dev.interactive(TRUE),
  id.n = 3,
  id.label = NULL,
  main = "{type} for {sQuote(name)}",
  xlab = NULL,
  ylim = NULL,
  cex.id = 0.75
)

## S3 method for class 'gofN'
autoplot(
  x,
  against = .fitted,
  which = 1:2,
  mappings = list(),
  geom_args = list(),
  id.n = 3,
  id.label = NULL
)
```

Arguments

x	a gofN object.
against	what the residuals should be plotted against. Note that different methods use different formats: see Details. Categorical (factor and ordered) values are visualised using boxplots, with ordered values also adding a smoothing line like the quantitative. Defaults to the fitted values.

which	which to plot (1 for residuals plot, 2 for $\sqrt{\lvert R_i \rvert}$ scale plot, and 3 for normal quantile-quantile plot).
col, pch, cex, bg	vector of values (wrapped in I()), network attribute, or a formula whose RHS gives an expression in terms of network attributes to plot against.
...	additional arguments to plot(), qqnorm(), and qqline(), and others.
ask	whether the user should be prompted between the plots.
id.n	maximum number of extreme points to label explicitly.
id.label	specification for how extreme points are to be labeled, defaulting to network's index in the combined network.
main	a template for the plots' titles; these use glue()'s templating, with {type} replaced with the type of plot and {name} replaced with the statistic.
xlab	horizontal axis label; defaults to a character representation of against.
ylim	vertical range for the plots, interpreted as in graphics::plot(); can be specified as a list with 3 elements, giving the range for the corresponding plot according to the plot numbers for the which= argument, and can be used to ensure that, e.g., diagnostic plots for different models are on the same scale.
cex.id	scaling factor for characters used to label extreme points; see plot.lm().
mappings	a named list of lists of mappings constructed by ggplot2::aes() overriding the defaults. See Details below.
geom_args	a named list of lists of arguments overriding the defaults for the individual geoms. See Details below.

Details

For the plot() method, against and id.label can be vectors of values (enclosed in I() to be used as is), a character string identifying a network attribute, or a formula whose RHS gives an expression in terms of network attributes to plot against. The against formula may also contain a .fitted variable which will be substituted with the fitted values.

For autoplot.gofN(), against and id.label are interpreted as expressions in terms of network attributes and values generated by augment.gofN(), included .fitted for the fitted values.

Value

autoplot.gofN() returns a list of ggplot objects that if printed render to diagnostic plots. If there is only one, the object itself is returned.

Customising autoplot.gofN()

autoplot.gofN() constructs the plots out of ggplot2::ggplot(), ggplot2::geom_point() (for numeric against), ggplot2::geom_boxplot() for categorical or ordinal against), and ggplot2::geom_smooth() (for numeric or ordinal against), and ggrepel::geom_text_repel(). Mappings and arguments passed through mappings and geom_args override the respective defaults. They may have elements default (for ggplot()), point (for geom_point() and geom_boxplot()), smooth (for geom_smooth), and text (for geom_text_repel()).

See Also

gofN() for examples, plot.lm(), graphics::plot() for regression diagnostic plots and their parameters.

---

snctrl *Statnet Control*

Description

A utility to facilitate argument completion of control lists, reexported from statnet.common.

Currently recognised control parameters

This list is updated as packages are loaded and unloaded.

Package ergm:

control.ergm drop, init, init.method, main.method, force.main, main.hessian, checkpoint, resume, MPLE.samplesize, init.MPLE.samplesize, MPLE.type, MPLE.maxit, MPLE.nonvar, MPLE.nonident, MPLE.nonident.tol, MPLE.covariance.samplesize, MPLE.covariance.method, MPLE.covariance.sim.burnin, MPLE.covariance.sim.interval, MPLE.check, MPLE.constraints.ignore, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.interval, MCMC.burnin, MCMC.samplesize, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burr MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.return.stats, MCMC.runtime.traceplot, MCMC.maxedges, MCMC.addto.se, MCMC.packagenames, SAN.maxit, SAN.nsteps.times, SAN, MCMLE.termination, MCMLE.maxit, MCMLE.conv.min.pval, MCMLE.confidence, MCMLE.confidence.boost, MCMLE.confidence.boo MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize.m obs.MCMC.samplesize, obs.MCMC.effectiveSize, obs.MCMC.interval.mul, obs.MCMC.interval, obs.MCMC.burnin.mul, obs.MCMC.burnin, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density, MCMLE.min.depfac, MCMLE.sampsize.boost.pow, MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac, MCMLE.metric, MCMLE.method, MCMLE.dampening, MCMLE.dampening.min.ess, MCMLE.dampening.level, MCMLE.steplength.margin, MCMLE.steplength, MCMLE.steplength.parallel, MCMLE.sequential, MCMLE.density.guard.min, MCMLE.density.guard, MCMLE.effectiveSize, obs.MCMLE.effectiveSize, MCMLE.interval, MCMLE.burnin, MCMLE.samplesize.per_theta, MCMLE.samplesize.min, MCMLE.samplesize, obs.MCMLE.samplesize.per_theta, obs.MCMLE.samplesize.min, obs.MCMLE.samplesize, obs.MCMLE.interval, obs.MCMLE.burnin, MCMLE.steplength.solver, MCMLE.last.boost, MCMLE.steplength.esteq, MCMLE.steplength.miss.sample, MCMLE.steplength.min, MCMLE.effectiveSize.interval_drop, MCMLE.save_intermediates, MCMLE.nonvar, MCMLE.nonident, MCMLE.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.min_iterations, SA.max_iterations, SA.phase3_n, SA.interval, SA.burnin, SA.samplesize, CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.nsteps, CD.multiplicity, CD.nsteps.obs, CD.multiplicity.obs, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.reltol, CD.metric, CD.method, CD.dampening, CD.dampening.min.ess, CD.dampening.level, CD.steplength.margin, CD.steplength, CD.adaptive.epsilon, CD.steplength.esteq, CD.steplength.miss.sample, CD.steplength.min, CD.steplength.parallel, CD.steplength.solver, loglik, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.bridge bridge.nsteps, bridge.target.se, bridge.bidirectional, drop, MCMC.burnin, MCMC.burnin.between, MCMC.interval, MCMC.samplesize, obs.MCMC.burnin, obs.MCMC.burnin.between, obs.MCMC.interval, obs.MCMC.samplesize, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.ergm.godfather term.options

control.ergm3 drop, init, init.method, main.method, force.main, main.hessian, checkpoint, resume, MPLE.samplesize, init.MPLE.samplesize, MPLE.type, MPLE.maxit, MPLE.nonvar, MPLE.nonident, MPLE.nonident.tol, MPLE.covariance.samplesize, MPLE.covariance.method, MPLE.covariance.sim.burnin, MPLE.covariance.sim.interval, MPLE.check, MPLE.constraints.ignore, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.interval, MCMC.burnin, MCMC.samplesize, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burr MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.return.stats, MCMC.runtime.traceplot, MCMC.maxedges, MCMC.addto.se, MCMC.packagenames, SAN.maxit, SAN.nsteps.times, SAN, MCMLE.termination, MCMLE.maxit, MCMLE.conv.min.pval, MCMLE.confidence, MCMLE.confidence.boost, MCMLE.confidence.boo MCMLE.confidence.boost.lag, MCMLE.NR.maxit, MCMLE.NR.reltol, obs.MCMC.mul, obs.MCMC.samplesize.m obs.MCMC.samplesize, obs.MCMC.effectiveSize, obs.MCMC.interval.mul, obs.MCMC.interval, obs.MCMC.burnin.mul, obs.MCMC.burnin, obs.MCMC.prop, obs.MCMC.prop.weights, obs.MCMC.prop.args, obs.MCMC.impute.min_informative, obs.MCMC.impute.default_density, MCMLE.min.depfac, MCMLE.sampsize.boost.pow, MCMLE.MCMC.precision, MCMLE.MCMC.max.ESS.frac, MCMLE.metric, MCMLE.method, MCMLE.dampening, MCMLE.dampening.min.ess, MCMLE.dampening.level, MCMLE.steplength.margin, MCMLE.steplength, MCMLE.steplength.parallel, MCMLE.sequential, MCMLE.density.guard.min, MCMLE.density.guard, MCMLE.effectiveSize, obs.MCMLE.effectiveSize, MCMLE.interval, MCMLE.burnin, MCMLE.samplesize.per_theta, MCMLE.samplesize.min, MCMLE.samplesize, obs.MCMLE.samplesize.per_theta, obs.MCMLE.samplesize.min, obs.MCMLE.samplesize, obs.MCMLE.interval, obs.MCMLE.burnin, MCMLE.steplength.solver, MCMLE.last.boost, MCMLE.steplength.esteq, MCMLE.steplength.miss.sample, MCMLE.steplength.min, MCMLE.effectiveSize.interval_drop, MCMLE.save_intermediates, MCMLE.nonvar, MCMLE.nonident, MCMLE.nonident.tol, SA.phase1_n, SA.initial_gain, SA.nsubphases, SA.min_iterations, SA.max_iterations, SA.phase3_n, SA.interval, SA.burnin, SA.samplesize, CD.samplesize.per_theta, obs.CD.samplesize.per_theta, CD.nsteps, CD.multiplicity, CD.nsteps.obs, CD.multiplicity.obs, CD.maxit, CD.conv.min.pval, CD.NR.maxit, CD.NR.reltol, CD.metric, CD.method, CD.dampening, CD.dampening.min.ess, CD.dampening.level, CD.steplength.margin, CD.steplength, CD.adaptive.epsilon, CD.steplength.esteq, CD.steplength.miss.sample, CD.steplength.min, CD.steplength.parallel, CD.steplength.solver, loglik, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.gof.ergm nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gof.formula nsim, MCMC.burnin, MCMC.interval, MCMC.batch, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.logLik.ergm bridge.nsteps, bridge.target.se, bridge.bidirectional, drop, MCMC.burnin, MCMC.interval, MCMC.samplesize, obs.MCMC.samplesize, obs.MCMC.interval, obs.MCMC.burnin, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, obs.MCMC.prop,

obs.MCMC.prop.weights, obs.MCMC.prop.args, MCMC.maxedges, MCMC.packagenames, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.san SAN.maxit, SAN.tau, SAN.invcov, SAN.invcov.diag, SAN.nsteps.alloc, SAN.nsteps, SAN.samplesize, SAN.prop, SAN.prop.weights, SAN.prop.args, SAN.packagenames, SAN.ignore.finite.offsets, term.options, seed, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.simulate MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxruns, MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.ergm MCMC.burnin, MCMC.interval, MCMC.scale, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.formula MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

control.simulate.formula.ergm MCMC.burnin, MCMC.interval, MCMC.prop, MCMC.prop.weights, MCMC.prop.args, MCMC.batch, MCMC.effectiveSize, MCMC.effectiveSize.damp, MCMC.effectiveSize.maxr MCMC.effectiveSize.burnin.pval, MCMC.effectiveSize.burnin.min, MCMC.effectiveSize.burnin.max, MCMC.effectiveSize.burnin.nmin, MCMC.effectiveSize.burnin.nmax, MCMC.effectiveSize.burnin.PC, MCMC.effectiveSize.burnin.scl, MCMC.effectiveSize.order.max, MCMC.maxedges, MCMC.packagenames, MCMC.runtime.traceplot, network.output, term.options, parallel, parallel.type, parallel.version.check, parallel.inherit.MT, ...

Package ergm.multi:

control.gofN nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

control.gofN.ergm nsim, obs.twostage, array.max, simulate, obs.simulate, parallel, parallel.type, parallel.version.check, parallel.inherit.MT

See Also

statnet.common::snctrl()

split.network *A* split() *method for* network::network *objects.*

Description

Split a network into subnetworks on a factor.

Usage

```
## S3 method for class 'network'
split(x, f, drop = FALSE, sep = ".", lex.order = FALSE, ...)
```

Arguments

x	a network::network object.
f, drop, sep, lex.order	see split(); note that f must have length equal to network.size(x).
...	additional arguments, currently unused.

Value

A network.list containing the networks. These networks will inherit all vertex and edge attributes, as well as relevant network attributes.

See Also

network::get.inducedSubgraph()

---

twostarL-ergmTerm *Multilayer two-star*

Description

This term adds one statistic to the model, equal to the number of cross-layer two-stars or two-paths in the network.

Usage

```
# binary: twostarL(Ls, type, distinct=TRUE)
```

Arguments

Ls	a list (constructed by list() or c() of two Layer Logic specifications (c.f. Layer Logic section in the Layer() documentation) specifying the layers of interest.
type	if the network is directed, an argument to determine which configurations are counted: "out" Number of configurations $(i \to j), (i \to k)$, where $(i \to j)$ is in logical layer Ls[[1]] and $(i \to k)$ is in logical layer Ls[[2]]. "in" Number of configurations $(j \to i), (k \to i)$, where $(j \to i)$ is in logical layer Ls[[1]] and $(k \to i)$ is in logical layer Ls[[2]]. "path" Number of configurations $(j \to i), (i \to k)$, where $(j \to i)$ is in logical layer Ls[[1]] and $(i \to k)$ is in logical layer Ls[[2]]. This argument is ignored for undirected networks.
distinct	if TRUE, j and k above are required to be distinct. That is, the constituent edges may not be coincident or reciprocal.

See Also

ergmTerm for index of model terms currently visible to the package.

Keywords: directed, layer-aware, undirected, binary

---

uncombine_network *Split up a network into a list of subgraphs*

Description

Given a network created by combine_networks(), uncombine_network() returns a list of networks, preserving attributes that can be preserved.

Usage

```
uncombine_network(
  nw,
  split.vattr = nw %n% ".blockID.vattr",
  names.vattr = nw %n% ".blockName.vattr",
  use.subnet.cache = FALSE
)
```

Arguments

nw	a network::network created by combine_networks().
split.vattr	name of the vertex attribute on which to split, defaulting to the value of the ".blockID.vattr" network attribute.

`names.vattr` optional name of the vertex attribute to use as network names in the output list, defaulting to the value of the `".blockName.vattr"` network attribute.

`use.subnet.cache` whether to use subnet cache if available; this is only safe to do if the network is *not* used for its edges but only for its vertex and network attributes.

Value

a list of `network::network`s containing subgraphs on `split.vattr`. In particular,

- their basic properties (directedness and bipartednes) are the same as those of the input network;
- vertex attributes are split;
- edge attributes are assigned to their respective edges in the returned networks.

If `split.vattr` is a vector, only the first element is used and it's "popped".

See Also

`split.network()`

Examples

```
data(samplk)

o1 <- combine_networks(list(samplk1, samplk2, samplk3))
image(as.matrix(o1))

o1 <- uncombine_network(o1)
```

---

upper_tri-ergmConstraint

Only dyads in the upper-triangle of the sociomatrix may be toggled

---

Description

For a directed network, only dyads (i, j) for which $i < j$ may be toggled. Optional argument `attr` controls which subgraphs are thus restricted.

Usage

```
# upper_tri(attr = NULL)
```

Arguments

`attr` a vertex attribute specification (see Specifying Vertex attributes and Levels (`?nodal_attributes`) for details.)

See Also

ergmConstraint for index of constraints and hints currently visible to the package.

Keywords: directed, dyad-independent

Index