The sna Package

January 10, 2005

Version 0.51-0

Date 2005-1-09

Title Tools for Social Network Analysis

Author Carter T. Butts <buttsc@uci.edu>

Maintainer Carter T. Butts <buttsc@uci.edu>

Depends R (&gt;= 1.2.0)

Suggests rgl

Description A range of tools for social network analysis, including node and graph-level indices, structural distance and covariance methods, structural equivalence detection, p* modeling, random graph generation, and 2D/3D network visualization.

License GPL Version 2 or newer

URL http://erzuli.ss.uci.edu/R.stuff

R topics documented:

add.isolates .................................................. 4
addisolates.deprecated .................................... 5
bbnam ......................................................... 6
bbnam.bf .................................................... 10
betweenness .................................................. 12
blockmodel ................................................... 13
blockmodel.expand ......................................... 15
bonpow ....................................................... 17
centralgraph ............................................... 19
centralization ............................................. 20
closeness ................................................... 22
component.dist ............................................. 23
components ................................................ 25
connectedness ............................................. 26
consensus ................................................... 27
cugtest ..................................................... 29
degree ....................................................... 31
diag.remove ............................................... 33
R topics documented:

dyad.census ................................................................. 34
efficiency .................................................................. 35
equiv.clust ................................................................. 36
eval.edgeperturbation ............................................... 38
evcent ....................................................................... 39
event2dichot ............................................................. 41
gapply ................................................................. 42
geclust.boxstats ..................................................... 43
gclust.centralgraph ................................................. 45
gcor ................................................................. 46
gcov ................................................................. 48
gden ................................................................. 49
gdist.plotdiff .......................................................... 50
gdist.plotstats ........................................................ 51
gdist.plotstats ........................................................ 51
geodist ............................................................... 53
gliop ................................................................. 54
gplot ................................................................. 56
gplot.arrow ........................................................... 59
gplot.layout ............................................................ 60
gplot.loop ............................................................... 64
gplot.target ............................................................ 65
gplot.vertex ............................................................ 67
gplot3d ................................................................. 68
gplot3d.arrow .......................................................... 70
gplot3d.layout .......................................................... 71
gplot3d.loop ............................................................. 74
graphcent ............................................................... 75
grecip ................................................................. 76
gscor ................................................................. 77
gscov ................................................................. 80
gtrans ................................................................. 82
gvectorize ............................................................... 83
hdist ................................................................. 84
hierarchy ............................................................... 86
infect ................................................................. 87
interval.graph .......................................................... 89
is.connected ........................................................... 91
is.isolate .............................................................. 92
isolates ................................................................. 93
lab.optimize ........................................................... 94
lnam ................................................................. 97
lower.tri.remove ..................................................... 100
lubness ............................................................... 101
make.stochastic ...................................................... 102
mutuality .............................................................. 103
netcancor ............................................................. 104
netlm ................................................................. 107
netlogit ............................................................... 108
npostpred ............................................................. 110
nties ................................................................. 111
numperm ............................................................. 112
plot.bbnam ............................................................ 113
R topics documented:

plot.blockmodel .................................................. 114
plot.cugtest ........................................................ 115
plot.equiv.clust .................................................. 116
plot.lnам ............................................................. 118
plot.qaptest ........................................................ 118
plot.sociomatrix ................................................... 119
potscalered.mcmc .................................................. 120
prestige .............................................................. 121
print.bayes.factor .................................................. 123
print.bbnam .......................................................... 124
print.blockmodel ................................................... 125
print.cugtest ........................................................ 125
print.lnам ............................................................ 126
print.netcancor .................................................... 127
print.netlm .......................................................... 127
print.netlogit ....................................................... 128
print.qaptest ....................................................... 129
print.summary.bayes.factor ..................................... 129
print.summary.bbnam .............................................. 130
print.summary.blockmodel ....................................... 131
print.summary.cugtest ............................................ 131
print.summary.lnам .............................................. 132
print.summary.netcancor ....................................... 133
print.summary.netlm ............................................. 133
print.summary.netlogit ....................................... 134
print.summary.qaptest ......................................... 135
pstar ................................................................. 135
qaptest .............................................................. 138
reachability .......................................................... 140
read.nos ............................................................ 142
rgnm ................................................................. 143
rgraph ............................................................... 144
rguman .............................................................. 146
rgws ................................................................. 147
rmperm .............................................................. 149
rperm ............................................................... 150
sdmat ................................................................. 151
sedist ............................................................... 153
sna-deprecated ..................................................... 155
sna-internal ......................................................... 155
sna/operators ....................................................... 156
sr2css ............................................................... 157
stackcount .......................................................... 158
stresscent ........................................................... 158
structdist ........................................................... 160
summary.bayes.factor .......................................... 162
summary.bbnam .................................................... 163
summary.blockmodel ............................................. 164
summary.cugtest .................................................. 164
summary.lnам ..................................................... 165
summary.netcancor .............................................. 166
summary.netlm ................................................... 166
Description

Adds \( n \) isolates to the adjacency matrix (or matrices) in \( \text{dat} \).

Usage

```r
add.isolates(dat, n)
```

Arguments

- **dat**: One or more adjacency matrices
- **n**: The number of isolates to add

Details

If \( \text{dat} \) contains more than one adjacency matrix, the \( n \) isolates are added to each member of \( \text{dat} \).

Value

The updated graph stack.

Note

Isolate addition is particularly useful when computing structural distances between graphs of different orders; see the above reference for details.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

`isolates`
Examples

```r
g <- rgraph(10, 5) # Produce some random graphs
dim(g) # Get the dimensions of g
g <- add.isolates(g, 2) # Add 2 isolates to each graph in g
dim(g) # Now examine g
g```

addisolates-deprecated

Add Isolates to a Graph (Deprecated)

Description

Adds `n` isolates to the adjacency matrix (or matrices) in `dat`. This function has been replaced by `add.isolates`, and should no longer be used.

Usage

```r
addisolates(dat, n)
```

Arguments

- `dat`: One or more adjacency matrices
- `n`: The number of isolates to add

Details

If `dat` contains more than one adjacency matrix, the `n` isolates are added to each member of `dat`.

Value

The updated graph stack.

Note

Isolate addition is particularly useful when computing structural distances between graphs of different orders; see the above reference for details.

Author(s)

Carter T. Butts (butts@uci.edu)

References

See Also

isolates

Examples

```r
# Produce some random graphs

z <- rgraph(10, 5)
# Get the dimensions of g

dim(z)

# Not run: z <- addisolates(z, 2)  # Add 2 isolates to each graph in g

dim(z)

# Now examine z

g
```
and hence the posterior draws are implicitly conditional on the observation pattern.

**model**

String containing the error model to use; options are “actor,” “pooled,” and “fixed”

... Arguments to be passed by `bbnam` to the particular model method

**nprior**

Network prior matrix. This must be a matrix of dimension n x n, containing the arc/edge priors for the criterion network. (E.g., `nprior[i,j]` gives the prior probability of i sending the relation to j in the criterion graph.) If no network prior is provided, an uninformative prior on the space of networks will be assumed (i.e., p(i->j)=0.5). Missing values are not allowed.

**em**

Probability of a false negative; this may be in the form of a single number, one number per observation slice, one number per (directed) dyad, or one number per dyadic observation (fixed model only)

**ep**

Probability of a false positive; this may be in the form of a single number, one number per observation slice, one number per (directed) dyad, or one number per dyadic observation (fixed model only)

**emprior**

Parameters for the (beta) false negative prior; these should be in the form of an (alpha,beta) pair for the pooled model, and of an n x 2 matrix of (alpha,beta) pairs for the actor model. If no `emprior` is given, an uninformative prior (1,1) will be assumed; note that this is usually inappropriate, as described below. Missing values are not allowed.

**epprior**

Parameters for the (beta) false positive prior; these should be in the form of an (alpha,beta) pair for the pooled model, and of an n x 2 matrix of (alpha,beta) pairs for the actor model. If no `epprior` is given, an uninformative prior (1,1) will be assumed; note that this is usually inappropriate, as described below. Missing values are not allowed.

**diag**

Boolean indicating whether loops (matrix diagonals) should be counted as data

**mode**

A string indicating whether the data in question forms a “graph” or a “digraph”

**reps**

Number of replicate chains for the Gibbs sampler (pooled and actor models only)

**draws**

Integer indicating the total number of draws to take from the posterior distribution. Draws are taken evenly from each replication (thus, the number of draws from a given chain is `draws/reps`), and are randomly reordered to minimize dependence associated with position in the chain.

**burntime**

Integer indicating the burn-in time for the Markov Chain. Each replication is iterated `burntime` times before taking draws (with these initial iterations being discarded); hence, one should realize that each increment to burntime increases execution time by a quantity proportional to `reps`. (pooled and actor models only)

**quiet**

Boolean indicating whether MCMC diagnostics should be displayed (pooled and actor models only)

**outmode**

“posterior” indicates that the exact posterior probability matrix for the criterion graph should be returned, otherwise draws from the joint posterior are returned instead (fixed model only)

**anames**

A vector of names for the actors (vertices) in the graph

**onames**

A vector of names for the observers (possibly the actors themselves) whose reports are contained in the CSS
compute.sqrtrhat

A boolean indicating whether or not Gelman et al.’s potential scale reduction measure (an MCMC convergence diagnostic) should be computed (pooled and actor models only)

Details

The bbnam models a set of network data as reflecting a series of (noisy) observations by a set of participant/observers regarding an uncertain criterion structure. Each observer is assumed to send false positives (i.e., reporting a tie when none exists in the criterion structure) with probability $e^+$, and false negatives (i.e., reporting that no tie exists when one does in fact exist in the criterion structure) with probability $e^-$. The criterion network itself is taken to be a Bernoulli (di)graph.

Note that the present model includes three variants:

1. Fixed error probabilities: Each edge is associated with a known pair of false negative/false positive error probabilities (provided by the researcher). In this case, the posterior for the criterion graph takes the form of a matrix of Bernoulli parameters, with each edge being independent conditional on the parameter matrix.

2. Pooled error probabilities: One pair of (uncertain) false negative/false positive error probabilities is assumed to hold for all observations. Here, we assume that the researcher’s prior information regarding these parameters can be expressed as a pair of Beta distributions, with the additional assumption of independence in the prior distribution. Note that error rates and edge probabilities are not independent in the joint posterior, but the posterior marginals take the form of Beta mixtures and Bernoulli parameters, respectively.

3. Per observer (“actor”) error probabilities: One pair of (uncertain) false negative/false positive error probabilities is assumed to hold for each observation slice. Again, we assume that prior knowledge can be expressed in terms of independent Beta distributions (along with the Bernoulli prior for the criterion graph) and the resulting posterior marginals are Beta mixtures and a Bernoulli graph. (Again, it should be noted that independence in the priors does not imply independence in the joint posterior!)

By default, the bbnam routine returns (approximately) independent draws from the joint posterior distribution, each draw yielding one realization of the criterion network and one collection of accuracy parameters (i.e., probabilities of false positives/negatives). This is accomplished via a Gibbs sampler in the case of the pooled/actor model, and by direct sampling for the fixed probability model. In the special case of the fixed probability model, it is also possible to obtain directly the posterior for the criterion graph (expressed as a matrix of Bernoulli parameters); this can be controlled by the outmode parameter.

As noted, the taking of posterior draws in the nontrivial case is accomplished via a Markov Chain Monte Carlo method, in particular the Gibbs sampler; the high dimensionality of the problem ($O(n^2 + 2n)$) tends to preclude more direct approaches. At present, chain burn-in is determined ex ante on a more or less arbitrary basis by specification of the burntime parameter. Eventually, a more systematic approach will be utilized. Note that insufficient burn-in will result in inaccurate posterior sampling, so it’s not wise to skimp on burn time where otherwise possible. Similarly, it is wise to employ more than one Markov Chain (set by reps), since it is possible for trajectories to become “trapped” in metastable regions of the state space. Number of draws per chain being equal, more replications are usually better than few; consult Gelman et al. for details. A useful measure of chain convergence, Gelman and Rubin’s potential scale reduction ($\sqrt{R}$), can be computed using the compute.sqrtrhat parameter. The potential scale reduction measure is an ANOVA-like comparison of within-chain versus between-chain variance; it approaches 1 (from above) as the chain converges, and longer burn-in times are strongly recommended for chains with scale reductions in excess of 1.1 or thereabouts.
Finally, a cautionary concerning prior distributions: it is important that the specified priors actually reflect the prior knowledge of the researcher; otherwise, the posterior will be inadequately informed. In particular, note that an uninformative prior on the accuracy probabilities implies that it is a priori equally probable that any given actor’s observations will be informative or negatively informative (i.e., that i observing j sending a tie to k reduces p(j->k)). This is a highly unrealistic assumption, and it will tend to produce posteriors which are bimodal (one mode being related to the “informative” solution, the other to the “negatively informative” solution). A more plausible but still fairly diffuse prior would be Beta(3,5), which reduces the prior probability of an actor’s being negatively informative to 0.16, and the prior probability of any given actor’s being more than 50% likely to make a particular error (on average) to around 0.22. (This prior also puts substantial mass near the 0.5 point, which would seem consonant with the BKS studies.) Butts (2003) discusses a number of issues related to choice of priors for the bbnam, and users should consult this reference if matters are unclear before defaulting to the uninformative solution.

Value

An object of class bbnam, containing the posterior draws. The components of the output are as follows:

- anames: A vector of actor names.
- draws: An integer containing the number of draws.
- em: A matrix containing the posterior draws for probability of producing false negatives, by actor.
- ep: A matrix containing the posterior draws for probability of producing false positives, by actor.
- nactors: An integer containing the number of actors.
- net: An array containing the posterior draws for the criterion network.
- reps: An integer indicating the number of replicate chains used by the Gibbs sampler.

Note

As indicated, the posterior draws are conditional on the observed data, and hence on the data collection mechanism if the collection design is non-ignorable. Complete data (e.g., a CSS) and random tie samples are examples of ignorable designs; see Gelman et al. for more information concerning ignorability.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

npostpred, event2dichot, bbnam

Examples

# Create some random data
g <- rgraph(5)
g.p <- 0.8 * g + 0.2 * (1 - g)
dat <- rgraph(5, 5, tprob = g.p)

# Define a network prior
pnet <- matrix(ncol = 5, nrow = 5)
pnet[, ,] <- 0.5

# Define em and ep priors
pem <- matrix(nrow = 5, ncol = 2)
pem[, 1] <- 3
pem[, 2] <- 5
pep <- matrix(nrow = 5, ncol = 2)
pep[, 1] <- 3
pep[, 2] <- 5

# Draw from the posterior
b <- bbnam.bf(dat, model = "actor", nprior = pnet, emprior = pem, epprior = pep,
              burntime = 100, draws = 100)

# Print a summary of the posterior draws
summary(b)

bbnam.bf

Estimate Bayes Factors for the bbnam

Description

This function uses monte carlo integration to estimate the BFs, and tests the fixed probability,
pooled, and pooled by actor models. (See bbnam for details.)

Usage

bbnam.bf(dat, nprior = matrix(rep(0.5, dim(dat)[1]^2),
nrow = dim(dat)[1], ncol = dim(dat)[1]), em.fp = 0.5, ep.fp = 0.5,
emprior.pooled = c(1, 1), epprior.pooled = c(1, 1),
emprior.actor = cbind(rep(1, dim(dat)[1]), rep(1, dim(dat)[1])),
epprior.actor = cbind(rep(1, dim(dat)[1]), rep(1, dim(dat)[1])),
diag = FALSE, mode = "digraph", reps = 1000)

Arguments

dat  Data array to be analyzed. This array must be of dimension m x n x n, where n is
[V(G)], the first dimension indexes the observer, the second indexes the sender of
the relation, and the third dimension indexes the recipient of the relation. (E.g.,
dat[i, j, k] == 1 implies that i observed j sending the relation in question to
k.) Note that only dichotomous data is supported at present, and missing values
are permitted; the data collection pattern, however, is assumed to be ignorable,
and hence the posterior inferences are implicitly conditional on the observation
pattern.
nprior Network prior matrix. This must be a matrix of dimension n x n, containing the arc/edge priors for the criterion network. (E.g., nprior[i,j] gives the prior probability of i sending the relation to j in the criterion graph.) If no network prior is provided, an uninformative prior on the space of networks will be assumed (i.e., p(i->j)=0.5). Missing values are not allowed.

em.fp Probability of false negatives for the fixed probability model
ep.fp Probability of false positives for the fixed probability model
em.prior.pooled (alpha,beta) pairs for the (beta) false negative prior under the pooled model
ep.prior.pooled (alpha,beta) pairs for the (beta) false positive prior under the pooled model
em.prior.actor Matrix of per observer (alpha,beta) pairs for the (beta) false negative prior under the per observer/actor model
ep.prior.actor Matrix of per observer (alpha,beta) pairs for the (beta) false negative prior under the per observer/actor model
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the criterion graph can contain loops. Diag is false by default.
mode String indicating the type of graph being evaluated. "Digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. Mode is set to "digraph" by default.
reps Number of monte carlo draws to take

Details
The bbnam model (detailed in the bbnam function help) is a fairly simple model for integrating informant reports regarding social network data. bbnam.bf computes Bayes Factors (integrated likelihood ratios) for the three error submodels of the bbnam: fixed error probabilities, pooled error probabilities, and per observer/actor error probabilities.

Value
An object of class bayes.factor.

Note
It is important to be aware that the model parameter priors are essential components of the models to be compared; inappropriate parameter priors will result in misleading Bayes Factors.

Author(s)
Carter T. Butts (buttsc@uci.edu)

References
betweenness

See Also
bbnam

Examples

```
betweenness
```

Compute the Betweenness Centrality Scores of Network Positions

Description

`betweenness` takes a graph stack (`dat`) and returns the betweenness centralities of positions within one graph (indicated by `nodes` and `g`, respectively). Depending on the specified mode, betweenness on directed or undirected geodesics will be returned; this function is compatible with `centralization`, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by `centralization` to normalize the observed centralization score).

Usage

```
betweenness(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph",
           diag=FALSE, tmaxdev=FALSE, cmode="directed",
           geodist.precomp=NULL, rescale=FALSE)
```

Arguments

- `dat` Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
- `g` Integer indicating the index of the graph for which centralities are to be calculated. By default, `g`=1.
- `nodes` List indicating which nodes are to be included in the calculation. By default, all nodes are included.
- `gmode` String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `gmode` is set to "digraph" by default.
- `diag` Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.
- `tmaxdev` Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, `tmaxdev`=FALSE.
- `cmode` String indicating the type of betweenness centrality being computed (directed or undirected geodesics).
- `geodist.precomp` A `geodist` object precomputed for the graph to be analyzed (optional)
- `rescale` If true, centrality scores are rescaled such that they sum to 1.
Details
The betweenness of a vertex, v, is given by

\[ C_B(v) = \sum_{i,j,k} \frac{g_{ijk}}{g_{ij}} \]

where \( g_{ijk} \) is the number of geodesics from i to k through j. Conceptually, high-betweenness vertices lie on a large number of non-redundant shortest paths between other vertices; they can thus be thought of as “bridges” or “boundary spanners.”

Value
A vector containing the betweenness scores.

Warning
Rescale may cause unexpected results if all actors have zero betweenness.

Note
Judicious use of `geodist.precomp` can save a great deal of time when computing multiple path-based indices on the same network.

Author(s)
Carter T. Butts (buttsc@uci.edu)

References

See Also
centralization

Examples

```r
g <- rgraph(10)  # Draw a random graph with 10 members
betweenness(g)  # Compute betweenness scores
```

Description
Given an equivalence clustering object and a graph stack, `blockmodel` will form a blockmodel of the graph stack based on the classes in question, using the specified block content type.
Usage

```r
blockmodel(dat, ec, k=NULL, h=NULL, block.content="density",
plabels=ec$plabels, glabels=ec$glabels, rlabels=NULL,
mode="digraph", diag=FALSE)
```

Arguments

- **dat**: A graph stack
- **ec**: An object of class `equiv.clust`
- **k**: The number of classes to form (using `cutree`)
- **h**: The height at which to split classes (using `cutree`)
- **block.content**: A string indicating block content type (see below)
- **plabels**: A vector of labels to be applied to the individual nodes
- **glabels**: A vector of labels to be applied to the graphs being modeled
- **rlabels**: A vector of labels to be applied to the (reduced) roles
- **mode**: A string indicating whether we are dealing with graphs or digraphs
- **diag**: A boolean indicating whether loops are permitted

Details

`blockmodel` forms its eponymous models by using `cutree` to cut an equivalence clustering in the fashion specified by `k` and `h`. After forming clusters (roles), the input graphs are reordered and blockmodel reduction is applied. Currently supported reductions are:

1. **density**: block density, computed as the mean value of the block
2. **meanrowsum**: mean row sums for the block
3. **meancolsum**: mean column sums for the block
4. **sum**: total block sum
5. **median**: median block value
6. **min**: minimum block value
7. **max**: maximum block value
8. **types**: semi-intelligent coding of blocks by "type." Currently recognized types are (in order of precedence) "NA" (i.e., blocks with no valid data), "null" (i.e., all values equal to zero), "complete" (i.e., all values equal to 1), "1 covered" (i.e., all rows/cols contain a 1), "1 row-covered" (i.e., all rows contain a 1), "1 col-covered" (i.e., all cols contain a 1), and "other" (i.e., none of the above).

Density or median-based reductions are probably the most interpretable for most conventional analyses, though type-based reduction can be useful in examining certain equivalence class hypotheses (e.g., 1 covered and null blocks can be used to infer regular equivalence classes). Once a given reduction is performed, the model can be analyzed and/or expansion can be used to generate new graphs based on the inferred role structure.

Value

An object of class `blockmodel`. 
blockmodel.expand

Author(s)
Carter T. Butts (buttsc@uci.edu)

References

See Also
equiv.clust, blockmodel.expand

Examples
#Create a random graph with _some_ edge structure
G.p<-sapply(runif(20,0,1),rep,20)  #Create a matrix of edge
#probabilities
G<-rgraph(20,tprob=G.p)  #Draw from a Bernoulli graph
#distribution

#Cluster based on structural equivalence
eq<-equiv.clust(G)

#Form a blockmodel with distance relaxation of 10
B<-blockmodel(G,eq,h=10)
plot(B)  #Plot it

blockmodel.expand Generate a Graph (or Stack) from a Given Blockmodel Using Particular Expansion Rules

Description
blockmodel.expand takes a blockmodel and an expansion vector, and expands the former by making copies of the vertices.

Usage
blockmodel.expand(b, ev, mode="digraph", diag=FALSE)

Arguments
b Blockmodel object
ev A vector indicating the number of copies to make of each class (respectively)
mode A string indicating whether the result should be a “graph” or “digraph”
diag A boolean indicating whether or not loops should be permitted
The primary use of blockmodel expansion is in generating test data from a blockmodeling hypothesis. Expansion is performed depending on the content type of the blockmodel; at present, only density is supported. For the density content type, expansion is performed by interpreting the inter-class density as an edge probability, and by drawing random graphs from the Bernoulli parameter matrix formed by expanding the density model. Thus, repeated calls to `blockmodel.expand` can be used to generate a sample for monte carlo null hypothesis tests under a Bernoulli graph model.

**Value**
An adjacency matrix, or stack thereof.

**Note**
Eventually, other content types will be supported.

**Author(s)**
Carter T. Butts (buttsc@uci.edu)

**References**

**See Also**
`blockmodel`

**Examples**
```r
#Create a random graph with _some_ edge structure
g.p<-sapply(runif(20,0,1),rep,20)  #Create a matrix of edge probabilities
#Probabilities
g<-rgraph(20,tprob=g.p)  #Draw from a Bernoulli graph distribution

#Cluster based on structural equivalence
eq<-equiv.clust(g)

#Form a blockmodel with distance relaxation of 15
b<-blockmodel(g,eq,h=15)

#Draw from an expanded density blockmodel
g.e<-blockmodel.expand(b,rep(2,length(b$rlabels)))  #Two of each class
#g.e
```
bonpow

Find Bonacich Power Centrality Scores of Network Positions

Description

bonpow takes a graph stack (dat) and returns the Bonacich power centralities of positions within one graph (indicated by nodes and g, respectively). The decay rate for power contributions is specified by exponent (1 by default). This function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).

Usage

bonpow(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, tmaxdev=FALSE, exponent=1, rescale=FALSE, tol=1e-07)

Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
g Integer indicating the index of the graph for which centralities are to be calculated. By default, g=1.
nodes List indicating which nodes are to be included in the calculation. By default, all nodes are included.
gmode String indicating which type of graph being evaluated. "Digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. This is currently ignored.
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
tmaxdev Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev=FALSE.
exponent Exponent (decay rate) for the Bonacich power centrality score; can be negative
rescale If true, centralities are rescaled such that they sum to 1.
tol Tolerance for near-singularities during matrix inversion (see solve)

Details

Bonacich’s power centrality measure is defined by $C_{BP} (\alpha, \beta) = \alpha (I - \beta A)^{-1} A 1$, where $\beta$ is an attenuation parameter (set here by exponent) and $A$ is the graph adjacency matrix. (The coefficient $\alpha$ acts as a scaling parameter, and is set here (following Bonacich (1987)) such that the sum of squared scores is equal to the number of vertices. This allows 1 to be used as a reference value for the “middle” of the centrality range.) When $\beta \rightarrow 1/\lambda_{A1}$ (the reciprocal of the largest eigenvalue of $A$), this is to within a constant multiple of the familiar eigenvector centrality score; for other values of $\beta$, the behavior of the measure is quite different. In particular, $\beta$ gives positive and negative weight to even and odd walks, respectively, as can be seen from the series expansion $C_{BP} (\alpha, \beta) = \alpha \sum_{k=0}^{\infty} \beta^k A^{k+1} 1$ which converges so long as $|\beta| < 1/\lambda_{A1}$. The magnitude of $\beta$ controls the influence of distant actors on ego’s centrality score, with larger magnitudes indicating slower rates of decay. (High rates, hence, imply a greater sensitivity to edge effects.)
Interpretively, the Bonacich power measure corresponds to the notion that the power of a vertex is recursively defined by the sum of the power of its alters. The nature of the recursion involved is then controlled by the power exponent: positive values imply that vertices become more powerful as their alters become more powerful (as occurs in cooperative relations), while negative values imply that vertices become more powerful only as their alters become weaker (as occurs in competitive or antagonistic relations). The magnitude of the exponent indicates the tendency of the effect to decay across long walks; higher magnitudes imply slower decay. One interesting feature of this measure is its relative instability to changes in exponent magnitude (particularly in the negative case). If your theory motivates use of this measure, you should be very careful to choose a decay parameter on a non-ad hoc basis.

**Value**

A vector containing the centrality scores.

**Warning**

Singular adjacency matrices cause no end of headaches for this algorithm; thus, the routine may fail in certain cases. This will be fixed when I get a better algorithm. bonpow will not symmetrize your data before extracting eigenvectors; don’t send this routine asymmetric matrices unless you really mean to do so.

**Note**

The theoretical maximum deviation used here is not obtained with the star network, in general. For positive exponents, at least, the symmetric maximum occurs for an empty graph with one complete dyad (the asymmetric maximum is generated by the outstar). UCINET V seems not to adjust for this fact, which can cause some oddities in their centralization scores (thus, don’t expect to get the same numbers with both packages).

**Author(s)**

Carter T. Butts (butts@uci.edu)

**References**


**See Also**

centralization, evcent

**Examples**

```r
#Generate some test data
dat<-rgraph(10,mode="graph")
#Compute Bonpow scores
bonpow(dat,exponent=1,tol=1e-20)
bonpow(dat,exponent=-1,tol=1e-20)
```
**centralgraph**

---

**Find the Central Graph of a Labeled Graph Set**

**Description**

Returns the central graph of a set of labeled graphs, i.e. that graph in which i->j iff i->j in >=50% of the graphs within the set. If `normalize==TRUE`, then the value of the i,jth edge is given as the proportion of graphs in which i->j.

**Usage**

```r
centralgraph(dat, normalize=FALSE)
```

**Arguments**

- `dat`: Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors.
- `normalize`: Boolean indicating whether the results should be normalized. The result of this is the "mean matrix". By default, `normalize==FALSE`.

**Details**

The central graph of a set of graphs S is that graph C which minimizes the sum of Hamming distances between C and G in S. As such, it turns out (for the dichotomous case, at least), to be analogous to both the mean and median for sets of graphs. The central graph is useful in a variety of contexts; see the references below for more details.

**Value**

A matrix containing the central graph (or mean matrix)

**Note**

0.5 is used as the cutoff value regardless of whether or not the data is dichotomous (as is tacitly assumed). The routine is unaffected by data type when `normalize==TRUE`.

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

`hdist`
Examples

#Generate some random graphs
dat<-rgraph(10,5)
#Find the central graph
cg<-centralgraph(dat)
#Plot the central graph
gplot(cg)
#Now, look at the mean matrix
cg<-centralgraph(dat,normalize=TRUE)
print(cg)

centralization

Find the Centralization of a Given Network, for Some Measure of Centrality

Description

Centralization returns the centralization GLI (graph-level index) for a given graph in dat, given a (node) centrality measure FUN. Centralization follows Freeman’s (1979) generalized definition of network centralization, and can be used with any properly defined centrality measure. This measure must be implemented separately; see the references below for examples.

Usage

centralization(dat, FUN, g=1, mode="digraph", diag=FALSE, normalize=TRUE, ...)

Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Provided that FUN is well-behaved, this can be an n x n matrix if only one graph is involved.
FUN Function to return nodal centrality scores.
g Integer indicating the index of the graph for which centralization should be computed. By default, g=1.
mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
normalize Boolean indicating whether or not the centralization score should be normalized to the theoretical maximum. (Note that this function relies on FUN to return this value when called with tmaxdev==TRUE.) By default, tmaxdev==TRUE.
... Additional arguments to FUN.
Details

The centralization of a graph $G$ for centrality measure $C(v)$ is defined (as per Freeman (1979)) to be:

$$C^*(G) = \sum_{i \in V(G)} \left| \max_{v \in V(G)} (C(v)) - C(i) \right|$$

Or, equivalently, the absolute deviation from the maximum of $C$ on $G$. Generally, this value is normalized by the theoretical maximum centralization score, conditional on $|V(G)|$. (Here, this functionality is activated by `normalize`.) Centralization depends on the function specified by `FUN` to return the vector of nodal centralities when called with `dat` and `g`, and to return the theoretical maximum value when called with the above and `tmaxdev==TRUE`. For an example of such a centrality routine, see `degree`.

Value

The centralization of the specified graph.

Note

See `cugtest` for null hypothesis tests involving centralization scores.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

`cugtest`

Examples

```r
# Generate some random graphs
dat<-rgraph(5,10)
# How centralized is the third one on indegree?
centralization(dat,g=3,degree,cmode="indegree")
# How about on total (Freeman) degree?
centralization(dat,g=3,degree)
```
closeness

Compute the Closeness Centrality Scores of Network Positions

Description

closeness takes a graph stack (dat) and returns the closeness centralities of positions within one graph (indicated by nodes and g, respectively). Depending on the specified mode, closeness on directed or undirected geodesics will be returned; this function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).

Usage

closeness(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, tmaxdev=FALSE, cmode="directed", geodist.precomp=NULL, rescale=FALSE)

Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
g Integer indicating the index of the graph for which centralities are to be calculated. By default, g=1.
nodes List indicating which nodes are to be included in the calculation. By default, all nodes are included.
gmode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. gmode is set to "digraph" by default.
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
tmaxdev Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.
cmode String indicating the type of closeness centrality being computed (distances on directed or undirected geodesics).
geodist.precomp A geodist object precomputed for the graph to be analyzed (optional)
rescale If true, centrality scores are rescaled such that they sum to 1.

Details

The closeness of a vertex v is defined as

$$C_C(v) = \frac{|V(G)| - 1}{\sum_{i \neq v} d(v, i)}$$

where \(d(i,j)\) is the geodesic distance between i and j (where defined). Closeness is ill-defined on disconnected graphs; in such cases, this routine substitutes \(\text{Inf}\). It should be understood that this modification is not canonical (though it is common), but can be avoided by not attempting to
measure closeness on disconnected graphs in the first place! Intuitively, closeness provides an index of the extent to which a given vertex has short paths to all other vertices in the graph; this is one reasonable measure of the extent to which a vertex is in the “middle” of a given structure.

Value

A vector containing the closeness scores.

Note

Judicious use of geodist.precomp can save a great deal of time when computing multiple path-based indices on the same network. Before sna version 0.50, closeness substituted one plus the maximum possible path length for disconnected vertex pairs.

Author(s)

Carter T. Butts, {butts@uci.edu}

References


See Also

centralization

Examples

g<-rgraph(10)  # Draw a random graph with 10 members
closeness(g)   # Compute closeness scores

---

calculate the Component Size Distribution of a Graph

Description

calculate returns a data frame containing a vector of length n such that the ith element contains the number of components of G having size i, and a vector of length n giving component membership. Component strength is determined by the connected parameter; see below for details.

Usage

calculate (dat, connected=c("strong","weak","unilateral","recursive"))

Arguments

dat A single nxn adjacency matrix
connected A string selecting strong, weak, unilateral or recursively connected components; by default, "strong" components are used.
Components are maximal sets of mutually connected vertices; depending on the definition of “connected” one employs, one can arrive at several types of components. Those supported here are as follows (in increasing order of restrictiveness):

1. weak: \( v_1 \) is connected to \( v_2 \) iff there exists a semi-path from \( v_1 \) to \( v_2 \) (i.e., a path in the weakly symmetrized graph)
2. unilateral: \( v_1 \) is connected to \( v_2 \) iff there exists a directed path from \( v_1 \) to \( v_2 \) or a directed path from \( v_2 \) to \( v_1 \)
3. strong: \( v_1 \) is connected to \( v_2 \) iff there exists a directed path from \( v_1 \) to \( v_2 \) and a directed path from \( v_2 \) to \( v_1 \)
4. recursive: \( v_1 \) is connected to \( v_2 \) iff there exists a vertex sequence \( v_a, \ldots, v_z \) such that \( v_1, v_a, \ldots, v_z, v_2 \) and \( v_2, v_z, \ldots, v_a, v_1 \) are directed paths

Note that the above definitions are distinct for directed graphs only; if \( \text{dat} \) is symmetric, then the \( \text{connected} \) parameter has no effect.

Value

A data frame containing:

- membership: A vector of component memberships, by vertex
- csize: A vector of component sizes, by component
- cdist: A vector of length \( |V(G)| \) with the (unnormalized) empirical distribution function of component sizes

Note

Unilaterally connected component partitions may not be well-defined, since it is possible for a given vertex to be unilaterally connected to two vertices which are not unilaterally connected with one another. Consider, for instance, the graph \( a \rightarrow b \leftarrow c \rightarrow d \). In this case, the maximal unilateral components are \( ab \) and \( bcd \), with vertex \( b \) properly belonging to both components. For such graphs, a unique partition of vertices by component does not exist, and we “solve” the problem by allocating each “problem vertex” to one of its components on an essentially arbitrary basis. (\( \text{component.dist} \) generates a warning when this occurs.) It is recommended that the unilateral option be avoided where possible.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

\( \text{components, symmetrize, reachability geodist} \)
### examples

```r
# Generate a sparse random graph
# Find weak components
g <- rgraph(20, tprob = 0.075)
cd <- component.dist(g, connected = "weak")
cd$membership  # Who's in what component?
cd$csize        # What are the component sizes?
plot(l:1:length(cd$cdist), cd$cdist/sum(cd$cdist), ylim = c(0, 1), type = "h")
# Find strong components
cd <- component.dist(g, connected = "strong")
cd$membership  # Who's in what component?
cd$csize        # What are the component sizes?
plot(l:1:length(cd$cdist), cd$cdist/sum(cd$cdist), ylim = c(0, 1), type = "h")
```

### components

Find the Number of (Maximal) Components Within a Given Graph

**Description**

Returns the number of components within `dat`, using the connectedness rule given in `connected`.

**Usage**

```r
components(dat, connected = "strong", comp.dist.precomp = NULL)
```

**Arguments**

- `dat`: An nxn adjacency matrix
- `connected`: The symmetrizing rule to be used by `symmetrize` before components are extracted
- `comp.dist.precomp`: A component size distribution object from `component.dist` (optional)

**Details**

The `connected` parameter corresponds to the `rule` parameter of `component.dist`. By default, `components` returns the number of strong components, but weak components can be returned if so desired. For symmetric matrices, this is obviously a moot point.

**Value**

The number of components of `dat`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)
connectedness

References

See Also
component.dist, symmetrize

Examples

g<-rgraph(20,tprob=0.05)  #Generate a sparse random graph

#Find weak components
components(g,connected="weak")

#Find strong components
components(g,connected="strong")

connectedness  Compute Graph Connectedness Scores

Description
connectedness takes a graph stack (dat) and returns the Krackhardt connectedness scores for the graphs selected by g.

Usage
connectedness(dat, g=1:stackcount(dat))

Arguments
dat  A graph or graph stack
g  Index values for the graphs to be utilized; by default, all graphs are selected

Details
Krackhardt’s connectedness for a digraph \(G\) is equal to the fraction of all dyads, \(\{i, j\}\), such that there exists an undirected path from \(i\) to \(j\) in \(G\). (This, in turn, is just the density of the weak reachability graph of \(G\).) Obviously, the connectedness score ranges from 0 (for the null graph) to 1 (for weakly connected graphs).

Connectedness is one of four measures (connectedness, efficiency, hierarchy, and lubness) suggested by Krackhardt for summarizing hierarchical structures. Each corresponds to one of four axioms which are necessary and sufficient for the structure in question to be an outtree; thus, the measures will be equal to 1 for a given graph iff that graph is an outtree. Deviations from unity can be interpreted in terms of failure to satisfy one or more of the outtree conditions, information which may be useful in classifying its structural properties.

Value
A vector containing the connectedness scores
Note

The four Krackhardt indices are, in general, nondegenerate for a relatively narrow band of size/density combinations (efficiency being the sole exception). This is primarily due to their dependence on the reachability graph, which tends to become complete rapidly as size/density increase. See Krackhardt (1994) for a useful simulation study.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

connectedness, efficiency, hierarchy, lubness, reachability

Examples

# Get connectedness scores for graphs of varying densities
connectedness(rgraph(10,5,tprob=c(0.1,0.25,0.5,0.75,0.9)))

consensus

Estimate a Consensus Structure from Multiple Observations

Description

consensus estimates a central or consensus structure given multiple observations, using one of several algorithms.

Usage

consensus(dat, mode="digraph", diag=FALSE, method="central.graph", tol=0.01)

Arguments

dat An m x n x n graph stack
mode "digraph" for directed data, else "graph"
diag A boolean indicating whether the diagonals (loops) should be treated as data
method One of "central.graph", "single.reweight", "PCA.reweight", "LAS.intersection", "LAS.union", "OR.row", or "OR.col"
tol Tolerance for the iterative reweighting algorithm (not currently supported)
Details

The term “consensus structure” is used by a number of authors to reflect a notion of shared or common perceptions of social structure among a set of observers. As there are many interpretations of what is meant by “consensus” (and as to how best to estimate it), several algorithms are employed here:

1. `central.graph`: Estimate the consensus structure using the central graph. This corresponds to a “median response” notion of consensus.

2. `single.reweight`: Estimate the consensus structure using subject responses, reweighted by mean graph correlation. This corresponds to an “expertise-weighted vote” notion of consensus.

3. `PCA.reweight`: Estimate the consensus using the (scores on the) first component of a network PCA. This corresponds to a “shared theme” or “common element” notion of consensus.

4. `LAS.intersection`: Estimate the consensus structure using the locally aggregated structure (intersection rule). In this model, an $i\rightarrow j$ edge exists iff $i$ and $j$ agree that it exists.

5. `LAS.union`: Estimate the consensus structure using the locally aggregated structure (union rule). In this model, an $i\rightarrow j$ edge exists iff $i$ or $j$ agree that it exists.

6. `OR.row`: Estimate the consensus structure using own report. Here, we take each informant’s outgoing tie reports to be correct.

7. `OR.col`: Estimate the consensus structure using own report. Here, we take each informant’s incoming tie reports to be correct.

Note that the reweighted algorithms are not dichotomized by default; since these return valued graphs, dichotomization may be desirable prior to use.

It should be noted that a model for estimating an underlying criterion structure from multiple informant reports is provided in `bbnam`; if your goal is to reconstruct an “objective” network from informant reports, this may prove more useful.

Value

An adjacency matrix representing the consensus structure

Note

Eventually, this routine will also support the (excellent) consensus methods of Romney and Batchelder; since these are similar in many respects to the `bbnam` model, users may wish to try this alternative for now.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

bbnam, centralgraph

Examples

```r
# Generate some test data
g <- rgraph(5)
g.pobs <- g*0.9+(1-g)*0.5
g.obs <- rgraph(5,5,tprob=g.pobs)

# Find some consensus structures
consensus(g.obs) # Central graph
consensus(g.obs, method="single.reweight") # Single reweighting
consensus(g.obs, method="PCA.reweight") # 1st component in network PCA
```

Description

`cugtest` tests an arbitrary GLI (computed on `dat` by `FUN`) against a conditional uniform graph null hypothesis, via Monte Carlo simulation of likelihood quantiles. Some variation in the nature of the conditioning is available; currently, conditioning only on size, conditioning jointly on size and estimated tie probability (via density), and conditioning jointly on size and (bootstrapped) edge value distributions are implemented. Note that fair amount of flexibility is possible regarding CUG tests on functions of GLIs (Anderson et al., 1999). See below for more details.

Usage

```r
cugtest(dat, FUN, reps=1000, gmode="digraph", cmode="density", diag=FALSE, g1=1, g2=2, ...)
```

Arguments

dat  Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Provided that `FUN` is well-behaved, this can be an n x n matrix if only one graph is involved.

FUN  Function to compute GLIs, or functions thereof. `FUN` must accept `dat` and the specified `g` arguments, and should return a real number.

reps  Integer indicating the number of draws to use for quantile estimation. Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, `reps==1000`.

gmode  String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `gmode` is set to "digraph" by default.
cmode  String indicating the type of conditioning assumed by the null hypothesis. If cmode is set to "density", then the density of the graph in question is used to determine the tie probabilities of the Bernoulli graph draws (which are also conditioned on |V(G)|). If cmode="ties", then draws are bootstrapped from the distribution of edge values within the data matrices. If cmode="order", then draws are uniform over all graphs of the same order (size) as the graphs within the input stack. By default, cmode is set to "density".

diag  Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

g1  Integer indicating the index of the first graph input to the GLI. By default, g1==1.

g2  Integer indicating the index of the second graph input to the GLI. (FUN can ignore this, if one wishes to test the GLI value of a single graph, but it should recognize the argument.) By default, g2==2.

...  Additional arguments to FUN.

Details

The null hypothesis of the CUG test is that the observed GLI (or function thereof) was drawn from a distribution isomorphic to that of said GLI evaluated (uniformly) on the space of all graphs conditional on one or more features. The most common "features" used for conditioning purposes are order (size) and density, both of which are known to have strong and nontrivial effects on other GLIs (Anderson et al., 1999) and which are, in many cases, exogenously determined. Since theoretical results regarding functions of arbitrary GLIs on the space of graphs are not available, the standard approach to CUG testing is to approximate the quantiles of the likelihood associated with the null hypothesis using Monte Carlo methods. This is the technique utilized by cugtest, which takes appropriately conditioned draws from the set of graphs and computes on them the GLI specified in FUN, thereby accumulating an approximation to the true likelihood.

The cugtest procedure returns a cugtest object containing the estimated likelihood (distribution of the test GLI under the null hypothesis), the observed GLI value of the data, and the one-tailed p-values (estimated quantiles) associated with said observation. As usual, the (upper tail) null hypothesis is rejected for significance level alpha if p>=observation is less than alpha (or p<=observation, for the lower tail); if the hypothesis is undirected, then one rejects if either p<=observation or p>=observation is less then alpha/2. Standard caveats regarding the use of null hypothesis testing procedures are relevant here: in particular, bear in mind that a significant result does not necessarily imply that the likelihood ratio of the null model and the alternative hypothesis favors the latter.

Informative and aesthetically pleasing portrayals of cugtest objects are available via the print.cugtest and summary.cugtest methods. The plot.cugtest method displays the estimated distribution, with a reference line signifying the observed value.

Value

An object of class cugtest, containing

testval  The observed GLI value.
dist    A vector containing the Monte Carlo draws.
pgeq    The proportion of draws which were greater than or equal to the observed GLI value.
pleq    The proportion of draws which were less than or equal to the observed GLI value.
**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

qaptest, gliop

**Examples**

```r
# Draw two random graphs, with different tie probabilities
dat<-rgraph(20,2,tprob=c(0.2,0.8))
# Is their correlation higher than would be expected, conditioning
# only on size?
cug<-cugtest(dat,gcor,cmode="order")
supply(cug)
# Now, let's try conditioning on density as well.
cug<-cugtest(dat,gcor)
supply(cug)
```

---

**degree**

*Compute the Degree Centrality Scores of Network Positions*

**Description**

Degree takes a graph stack (dat) and returns the degree centralities of positions within one graph (indicated by nodes and g, respectively). Depending on the specified mode, indegree, outdegree, or total (Freeman) degree will be returned; this function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).

**Usage**

```r
degree(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, tmaxdev=FALSE, cmode="freeman", rescale=FALSE)
```

**Arguments**

- **dat**
  Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Provided that FUN is well-behaved, this can be an n x n matrix if only one graph is involved.

- **g**
  Integer indicating the index of the graph for which centralities are to be calculated. By default, g=1.

- **nodes**
  List indicating which nodes are to be included in the calculation. By default, all nodes are included.

- **gmode**
  String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. gmode is set to "digraph" by default.
degree

diag  
Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

tmaxdev  
Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.

cmode  
String indicating the type of degree centrality being computed. "indegree", "outdegree", and "freeman" refer to the indegree, outdegree, and total (Freeman) degree measures, respectively. The default for cmode is "freeman".

rescale  
If true, centrality scores are rescaled such that they sum to 1.

Details

Degree centrality is the social networker’s term for various permutations of the graph theoretic notion of vertex degree: indegree of a vertex, \( v \), corresponds to the cardinality of the vertex set \( N^+(v) = \{ i \in V(G) : (i, v) \in E(G) \} \); outdegree corresponds to the cardinality of the vertex set \( N^-(v) = \{ i \in V(G) : (v, i) \in E(G) \} \); and total (or "Freeman") degree corresponds to \(|N^+(v)| + |N^-(v)|\). (Note that, for simple graphs, indegree=outdegree=total degree/2.) Obviously, degree centrality can be interpreted in terms of the sizes of actors’ neighborhoods within the larger structure. See the references below for more details.

Value

A vector containing the degree centrality scores

Author(s)

Carter T. Butts (butsc@uci.edu)

References


See Also

centralization

Examples

#Create a random directed graph
dat<-rgraph(10)
#Find the indegrees, outdegrees, and total degrees
degree(dat,cmode="indegree")
degree(dat,cmode="outdegree")
degree(dat)
diag.remove

Remove the Diagonals of Adjacency Matrices in a Graph Stack

Description

Returns the input graph stack, with the diagonal entries removed/replaced as indicated.

Usage

diag.remove(dat, remove.val=NA)

Arguments

- **dat**: A graph stack
- **remove.val**: The value with which to replace the existing diagonals

Details

diag.remove is simply a convenient way to apply diag to an entire stack of adjacency matrices at once.

Value

The updated graph stack

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

diag.upper.tri.remove, lower.tri.remove

Examples

```r
# Generate a random graph stack
g <- rgraph(3, 5)
# Remove the diagonals
g <- diag.remove(g)
```
## dyad.census

**Compute a Holland and Leinhardt MAN Dyad Census**

### Description

dyad.census computes a Holland and Leinhardt dyad census on the graphs of `dat` selected by `g`.

### Usage

dyad.census(dat, g=1:stackcount(dat))

### Arguments

- **dat**: A graph or graph stack
- **g**: The elements of `dat` to be included; by default, all graphs are processed.

### Details

Each dyad in a directed graph may be in one of four states: the null state \((a \not\leftrightarrow b)\), the complete or mutual state \((a \leftrightarrow b)\), and either of two asymmetric states \((a \leftarrow b)\) or \((a \rightarrow b)\). Holland and Leinhardt’s dyad census classifies each dyad into the mutual, asymmetric, or null categories, counting the number of each within the digraph. These counts can be used as the basis for null hypothesis tests (since their distributions are known under assumptions such as constant edge probability), or for the generation of random graphs (e.g., via the U|MAN distribution, which conditions on the numbers of mutual, asymmetric, and null dyads in each graph).

### Value

A matrix whose three columns contain the counts of mutual, asymmetric, and null dyads (respectively) for each graph.

### Author(s)

Carter T. Butts (buttsc@uci.edu)

### References


### See Also

- `mutuality`, `grecip`, `rguman`, `triad.census`

### Examples

```r
# Generate a dyad census of random data with varying densities
dyad.census(rgraph(15,5,tprob=c(0.1,0.25,0.5,0.75,0.9)))
```
Compute Graph Efficiency Scores

Description

`efficiency` takes a graph stack (`dat`) and returns the Krackhardt efficiency scores for the graphs selected by `g`.

Usage

```r
efficiency(dat, g=1:stackcount(dat), diag=FALSE)
```

Arguments

- `dat`: A graph or graph stack
- `g`: Index values for the graphs to be utilized; by default, all graphs are selected
- `diag`: TRUE if the diagonal contains valid data; by default, `diag==FALSE`

Details

Let $G = \bigcup_{i=1}^{n} G_i$ be a digraph with weak components $G_1, G_2, \ldots, G_n$. For convenience, we denote the cardinalities of these components’ vertex sets by $|V(G)| = N$ and $|V(G_i)| = N_i$, $\forall i \in 1, \ldots, n$. Then the Krackhardt efficiency of $G$ is given by

$$1 - \frac{|E(G)| - \sum_{i=1}^{n} (N_i - 1)}{\sum_{i=1}^{n} (N_i (N_i - 1) - (N_i - 1))}$$

which can be interpreted as $1$ minus the proportion of possible “extra” edges (above those needed to weakly connect the existing components) actually present in the graph. A graph which an efficiency of $1$ has precisely as many edges as are needed to connect its components; as additional edges are added, efficiency gradually falls towards $0$.

Efficiency is one of four measures (`connectedness`, `efficiency`, `hierarchy`, and `lubness`) suggested by Krackhardt for summarizing hierarchical structures. Each corresponds to one of four axioms which are necessary and sufficient for the structure in question to be an outtree; thus, the measures will be equal to $1$ for a given graph iff that graph is an outtree. Deviations from unity can be interpreted in terms of failure to satisfy one or more of the outtree conditions, information which may be useful in classifying its structural properties.

Value

A vector of efficiency scores

Note

The four Krackhardt indices are, in general, nondegenerate for a relatively narrow band of size/density combinations (efficiency being the sole exception). This is primarily due to their dependence on the reachability graph, which tends to become complete rapidly as size/density increase. See Krackhardt (1994) for a useful simulation study.

The violation normalization used before version 0.51 was $N (N - 1) \sum_{i=1}^{n} (N_i - 1)$, based on a somewhat different interpretation of the definition in Krackhardt (1994). The former version gave results which more closely matched those of the cited simulation study, but was less consistent with the textual definition.
equiv.clust

Find Clusters of Positions Based on an Equivalence Relation

description

equiv.clust uses a definition of approximate equivalence (equiv.fun) to form a hierarchical clustering of network positions. Where dat consists of multiple relations, all specified relations are considered jointly in forming the equivalence clustering.

Usage

equiv.clust(dat, g=c(1:dim(dat)[1]), equiv.fun="sedist", method="hamming", mode="digraph", diag=FALSE, cluster.method="complete", glabels=dimnames(dat)[[1]][g], plabels=dimnames(dat)[[2]], ...)

Arguments

dat A graph adjacency matrix, or stack thereof
g The elements of dat to use in clustering the vertices; by default, all structures are used
equiv.fun The distance function to use in clustering vertices (defaults to sedist)
method method parameter to be passed to equiv.fun
mode "graph" or "digraph," as appropriate
diag A boolean indicating whether or not matrix diagonals (loops) should be interpreted as useful data
cluster.method The hierarchical clustering method to use (see hclust)
glabels Labels for the various graphs in dat
plabels Labels for the vertices of dat
... Additional arguments to equiv.dist

Examples

#Get efficiency scores for graphs of varying densities
efficiency(rgraph(10,5,tprob=c(0.1,0.25,0.5,0.75,0.9)))
equiv.clust

Details

This routine is essentially a joint front-end to hclust and various positional distance functions, though it defaults to structural equivalence in particular. Taking the specified graphs as input, equiv.clust computes the distances between all pairs of positions using equiv.dist, and then performs a cluster analysis of the result. The return value is an object of class equiv.clust, for which various secondary analysis methods exist.

Value

An object of class equiv.clust

Note

See sedist for an example of a distance function compatible with equiv.clust.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

sedist, blockmodel

Examples

#Create a random graph with _some_ edge structure
g.p<-sapply(runif(20,0,1),rep,20) #Create a matrix of edge probabilities
g<-rgraph(20,tprob=g.p) #Draw from a Bernoulli graph distribution

#Cluster based on structural equivalence
eq<-equiv.clust(g)
plot(eq)
eval.edgeperturbation
Compute the Effects of Single-Edge Perturbations on Structural Indices

Description
Evaluates a given function on an input graph with and without a specified edge, returning the difference between the results in each case.

Usage
eval.edgeperturbation(dat, i, j, FUN, ...)

Arguments
dat A single adjacency matrix
i The row(s) of the edge(s) to be perturbed
j The column(s) of the edge(s) to be perturbed
FUN The function to be computed
... Additional arguments to FUN

Details
Although primarily a back-end utility for pstar, eval.edgeperturbation may be useful in any circumstance in which one wishes to assess the stability of a given structural index with respect to single edge perturbations. The function to be evaluated is calculated first on the input graph with all marked edges set to present, and then on the same graph with said edges absent. (Obviously, this is sensible only for dichotomous data.) The difference is then returned.
In pstar, calls to eval.edgeperturbation are used to construct a perturbation effect matrix for the GLM.

Value
The difference in the values of FUN as computed on the perturbed graphs.

Note
length(i) and length(j) must be equal; where multiple edges are specified, the row and column listings are interpreted as pairs.

Author(s)
Carter T. Butts (butts@uci.edu)

References
evcent

See Also
pstar

Examples

# Create a random graph
g<-rgraph(5)

# How much does a one-edge change affect reciprocity?
eval.edgeperturbation(g,1,2,grecip)

evcent

Find Eigenvector Centrality Scores of Network Positions

Description

evcent takes a graph stack (dat) and returns the eigenvector centralities of positions within one graph (indicated by nodes and g, respectively). This function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).

Usage

evcent(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, tmaxdev=FALSE, rescale=FALSE)

Arguments

dat
Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).

g
Integer indicating the index of the graph for which centralities are to be calculated. By default, g==1.

nodes
List indicating which nodes are to be included in the calculation. By default, all nodes are included.

gmode
String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. This is currently ignored.

diag
Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

tmaxdev
Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.

rescale
If true, centrality scores are rescaled such that they sum to 1.
Eigenvector centrality scores correspond to the values of the first eigenvector of the graph adjacency matrix; these scores may, in turn, be interpreted as arising from a reciprocal process in which the centrality of each actor is proportional to the sum of the centralities of those actors to whom he or she is connected. In general, vertices with high eigenvector centralities are those which are connected to many other vertices which are, in turn, connected to many others (and so on). (The perceptive may realize that this implies that the largest values will be obtained by individuals in large cliques (or high-density substructures). This is also intelligible from an algebraic point of view, with the first eigenvector being closely related to the best rank-1 approximation of the adjacency matrix (a relationship which is easy to see in the special case of a diagonalizable symmetric real matrix via the $SA S^{-1}$ decomposition).)

The simple eigenvector centrality is generalized by the Bonacich power centrality measure; see `bonpow` for more details.

A vector containing the centrality scores

`evcent` will not symmetrize your data before extracting eigenvectors; don’t send this routine asymmetric matrices unless you really mean to do so.

The theoretical maximum deviation used here is not obtained with the star network, in general. For symmetric data, the maximum occurs for an empty graph with one complete dyad; the maximum deviation for asymmetric data is generated by the outstar. UCINET V seems not to adjust for this fact, which can cause some oddities in their centralization scores (and results in a discrepancy in centralizations between the two packages).

Carter T. Butts ⟨butsc@uci.edu⟩


See Also

`centralization`, `bonpow`

Examples

```r
#Generate some test data
dat<-rgraph(10,mode="graph")
#Compute eigenvector centrality scores
evcent(dat)
```
event2dichot

Convert an Observed Event Matrix to a Dichotomous matrix

Description

Given a valued adjacency matrix (possibly derived from observed interaction “events”), event2dichot returns a dichotomous adjacency matrix.

Usage

event2dichot(m, method="quantile", thresh=0.5, leq=FALSE)

Arguments

m
An adjacency matrix

method
One of “quantile,” “rquantile,” “cquantile,” “mean,” “rmean,” “cmean,” “absolute,” “rank,” “rrank,” or “crank”

thresh
Dichotomization thresholds for ranks or quantiles

leq
Boolean indicating whether values less than or equal to the threshold should be taken as existing edges; the alternative is to use values strictly greater than the threshold

Details

The methods used for choosing dichotomization thresholds are as follows:

1. quantile: specified quantile over the distribution of all edge values
2. rquantile: specified quantile by row
3. cquantile: specified quantile by column
4. mean: grand mean
5. rmean: row mean
6. cmean: column mean
7. absolute: the value of thresh itself
8. rank: specified rank over the distribution of all edge values
9. rrank: specified rank by row
10. crank: specified rank by column

Note that when a quantile, rank, or value is said to be “specified,” this refers to the value of thresh.

Value

The dichotomized data matrix

Author(s)

Carter T. Butts (<buttsc@uci.edu>)
References


Examples

```r
# Draw a matrix of normal values
n <- matrix(rnorm(25), nrow = 5, ncol = 5)

# Dichotomize by the mean value
event2dichot(n, "mean")

# Dichotomize by the 0.95 quantile
event2dichot(n, "quantile", 0.95)
```

---

**gapply**

*Apply Functions Over Vertex Neighborhoods*

**Description**

Returns a vector or array or list of values obtained by applying a function to vertex neighborhoods of a given order.

**Usage**

```r
gapply(X, MARGIN, STATS, FUN, ..., mode = "digraph", diag = FALSE, distance = 1, thresh = 0, simplify = TRUE)
```

**Arguments**

- **X**
  - the adjacency matrix to be used.

- **MARGIN**
  - a vector giving the "margin" of X to be used in calculating neighborhoods. 1 indicates rows (out-neighbors), 2 indicates columns (in-neighbors), and c(1,2) indicates rows and columns (total neighborhood).

- **STATS**
  - the vector or matrix of vertex statistics to be used.

- **FUN**
  - the function to be applied. In the case of operators, the function name must be quoted.

- **...**
  - additional arguments to FUN.

- **mode**
  - "graph" if X is a simple graph, else "digraph".

- **diag**
  - boolean; are the diagonals of X meaningful?

- **distance**
  - the maximum geodesic distance at which neighborhoods are to be taken. 1 signifies first-order neighborhoods, 2 signifies second-order neighborhoods, etc.

- **thresh**
  - the threshold to be used in dichotomizing X.

- **simplify**
  - boolean; should we attempt to coerce output to a vector if possible?
Details

For each vertex in \( X \), \texttt{gapply} first identifies all members of the relevant neighborhood (as determined by \texttt{MARGIN} and \texttt{distance}) and pulls the rows of \texttt{STATS} associated with each. \texttt{FUN} is then applied to this collection of values. This provides a very quick and easy way to answer questions like:

- How many persons are in each ego’s 3rd-order neighborhood?
- What fraction of each ego’s alters are female?
- What is the mean income for each ego’s trading partners?
- etc.

With clever use of \texttt{FUN} and \texttt{STATS}, a wide range of functionality can be obtained.

Value

The result of the iterated application of \texttt{FUN} to each vertex neighborhood’s \texttt{STATS}.

Author(s)

Carter T. Butts \( \langle \text{buttsc@uci.edu} \rangle \)

See Also

\texttt{apply}, \texttt{sapply}

Examples

```r
#Generate a random graph
g<-rgraph(6)

#Calculate the degree of g using gapply
all(gapply(g,1,rep(1,6),sum)==degree(g,cmode="outdegree"))
all(gapply(g,2,rep(1,6),sum)==degree(g,cmode="degree"))
all(gapply(g,c(1,2),rep(1,6),sum)==degree(symmetrize(g),cmode="freeman")/2)

#Find first and second order neighborhood means on some variable
gapply(g,c(1,2),1:6,mean)
gapply(g,c(1,2),1:6,mean,distance=2)
```

---

\texttt{gclust.boxstats} \hspace{1cm} \textit{Plot Statistics Associated with Graph Clusters}

Description

\texttt{gclust.boxstats} creates side-by-side boxplots of graph statistics based on a hierarchical clustering of networks (cut into \( k \) sets).

Usage

\texttt{gclust.boxstats(h, k, meas, ...)}
gclust.boxstats

Arguments

h An `hclust` object, presumably formed by clustering a set of structural distances
k The number of groups to evaluate
meas A vector of length equal to the number of graphs in `h`, containing a GLI to be evaluated
... Additional parameters to `boxplot`

Details

gclust.boxstats simply takes the `hclust` object in `h`, applies `cutree` to form `k` groups, and then uses `boxplot` on the distribution of `meas` by group. This can be quite handy for assessing graph clusters.

Value

None

Note

Actually, this function will work with any `hclust` object and measure matrix; the data need not originate with social networks. For this reason, the clever may also employ this function in conjunction with `sedist` or `equiv.clust` to plot NLIs against clusters of positions within a graph.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

gclust.centralgraph, gdist.plotdiff, gdist.plotstats

Examples

#Create some random graphs
g<-rgraph(10,20,tprob=c(rbmeta(10,15,2),rbeta(10,2,15)))

#Find the Hamming distances between them
g.h<-hdist(g)

#Cluster the graphs via their Hamming distances
g.c<-hclust(as.dist(g.h))

#Now display boxplots of density by cluster for a two cluster solution
gclust.boxstats(g.c,2,gden(g))
gclust.centralgraph

*Get Central Graphs Associated with Graph Clusters*

**Description**

Calculates central graphs associated with particular graph clusters (as indicated by the $k$ partition of $h$).

**Usage**

```
gclust.centralgraph(h, k, mats, ...)```

**Arguments**

- **h**: An `hclust` object, the based on a graph set
- **k**: The number of groups to evaluate
- **mats**: A graph stack containing the adjacency matrices for the graphs on which the clustering was performed
- **...**: Additional arguments to `centralgraph`

**Details**

`gclust.centralgraph` uses `cutree` to cut the hierarchical clustering in $h$ into $k$ groups. `centralgraph` is then called on each cluster, and the results are returned as a graph stack. This is a useful tool for interpreting clusters of (labeled) graphs, with the resulting central graphs being subsequently analyzed using standard SNA methods.

**Value**

An array containing the stack of central graph adjacency matrices

**Note**

**Author(s)**

Carter T. Butts (butts@uci.edu)

**References**


**See Also**

`hclust, centralgraph, gclust.boxstats, gdist.plotdiff, gdist.plotstats`
Examples

```r
# Create some random graphs
g <- rgraph(10, 20, tprob = c(rbeta(10, 15, 2), rbeta(10, 2, 15)))

# Find the Hamming distances between them
g.h <- hdist(g)

# Cluster the graphs via their Hamming distances
g.c <- hclust(as.dist(g.h))

# Now find central graphs by cluster for a two cluster solution
g.cg <- gclust.centralgraph(g.c, 2, g)

# Plot the central graphs
gplot(g.cg[, 1, ])
gplot(g.cg[, 2, ])
```

---

gcor  
Find the (Product-Moment) Correlation Between Two or More Labeled Graphs

Description

gcor finds the product-moment correlation between the adjacency matrices of graphs indicated by g1 and g2 in stack dat (or possibly dat2). Missing values are permitted.

Usage

```r
gcor(dat, dat2 = NULL, g1 = c(1:dim(dat)[1]), g2 = c(1:dim(dat)[1]),
     diag = FALSE, mode = "digraph")
```

Arguments

dat  
A graph stack
dat2  
Optionally, a second graph stack
g1  
The indices of dat reflecting the first set of graphs to be compared; by default, all members of dat are included
g2  
The indices or dat (or dat2, if applicable) reflecting the second set of graphs to be compared; by default, all members of dat are included
diag  
Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
mode  
String indicating the type of graph being evaluated. "Digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.
The (product moment) graph correlation between labeled graphs G and H is given by

\[ cor(G, H) = \frac{cov(G, H)}{\sqrt{cov(G, G) cov(H, H)}} \]

where the graph covariance is defined as

\[ cov(G, H) = \frac{1}{(|V|)^2} \sum_{i,j} (A^G_{ij} - \mu_G)(A^H_{ij} - \mu_H) \]

(with \( A^G \) being the adjacency matrix of G). The graph correlation/covariance is at the center of a number of graph comparison methods, including network variants of regression analysis, PCA, CCA, and the like.

Note that \texttt{gcor} computes only the correlation between uniquely labeled graphs. For the more general case, \texttt{gscor} is recommended.

Value

A graph correlation matrix

Note

The \texttt{gcor} routine is really just a front-end to the standard \texttt{cor} method; the primary value-added is the transparent vectorization of the input graphs (with intelligent handling of simple versus directed graphs, diagonals, etc.). As noted, the correlation coefficient returned is a standard Pearson’s product-moment coefficient, and output should be interpreted accordingly. Classical null hypothesis testing procedures are not recommended for use with graph correlations; for nonparametric null hypothesis testing regarding graph correlations, see \texttt{cugtest} and \texttt{qaptest}. For multivariate correlations among graph sets, try \texttt{netcancor}.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

\texttt{gscor}, \texttt{gcov}, \texttt{gscov}

Examples

```R
#Generate two random graphs each of low, medium, and high density
g<-rgraph(10,6,tprob=c(0.2,0.2,0.5,0.5,0.8,0.8))

#Examine the correlation matrix
gcor(g)
```
Find the Covariance(s) Between Two or More Labeled Graphs

gcov

Description

gcov finds the covariances between the adjacency matrices of graphs indicated by g1 and g2 in stack dat (or possibly dat2). Missing values are permitted.

Usage

gcov(dat, dat2=NULL, g1=c(1:dim(dat)[1]), g2=c(1:dim(dat)[1]), diag=FALSE, mode="digraph")

Arguments

dat A graph stack
dat2 Optionally, a second graph stack
g1 The indices of dat reflecting the first set of graphs to be compared; by default, all members of dat are included
g2 The indices or dat (or dat2, if applicable) reflecting the second set of graphs to be compared; by default, all members of dat are included
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.

Details

The graph covariance between two labeled graphs is defined as

\[ \text{cov}(G, H) = \frac{1}{(\vert V \vert)^2} \sum_{(i,j)} \left( A^G_{ij} - \mu_G \right) \left( A^H_{ij} - \mu_H \right) \]

(with \( A^G \) being the adjacency matrix of G). The graph correlation/covariance is at the center of a number of graph comparison methods, including network variants of regression analysis, PCA, CCA, and the like.

Note that gcov computes only the covariance between uniquely labeled graphs. For the more general case, gscov is recommended.

Value

A graph covariance matrix

Note

The gcov routine is really just a front-end to the standard cov method; the primary value-added is the transparent vectorization of the input graphs (with intelligent handling of simple versus directed graphs, diagonals, etc.). Classical null hypothesis testing procedures are not recommended for use with graph covariance; for nonparametric null hypothesis testing regarding graph covariance, see cugtest and qaptest.
Find the Density of a Graph

**Description**

gden computes the density of graph \( g \) in stack \( \text{dat} \), adjusting for the type of graph in question.

**Usage**

gden(dat, g=NULL, diag=FALSE, mode="digraph")

**Arguments**

- **dat**
  Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. If \( \text{dat} \) is a matrix, \( g \) will be ignored.

- **g**
  Integer indicating the index of the graphs for which the density is to be calculated. If \( g=\text{NULL} \) (the default), density is calculated for all graphs in \( \text{dat} \).

- **diag**
  Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. \( \text{diag} \) is FALSE by default.

- **mode**
  String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. \( \text{mode} \) is set to "digraph" by default.

**Details**

The density of a graph is here taken to be the sum of tie values divided by the number of possible ties (i.e., an unbiased estimator of the graph mean); hence, the result is interpretable for valued graphs as the mean tie value. The number of possible ties is determined by the graph type (and by \( \text{diag} \)) in the usual fashion.
Value

The graph density

Author(s)

Carter T. Butts (butts@uci.edu)

References


Examples

```r
# Draw three random graphs
dat<-rgraph(10,3)
# Find their densities
gden(dat)
```

---

**gdist.plotdiff**

*Plot Differences in Graph-level Statistics Against Inter-graph Distances*

Description

For a given graph set, `gdist.plotdiff` plots the distances between graphs against their distances (or differences) on a set of graph-level measures.

Usage

```r
gdist.plotdiff(d, meas, method="manhattan", jitter=TRUE, xlab="Inter-Graph Distance", ylab="Measure Distance", lm.line=FALSE, ...)
```

Arguments

- `d` A matrix containing the inter-graph distances
- `meas` An n x m matrix containing the graph-level indices; rows of this matrix must correspond to graphs, and columns to indices
- `method` The distance method used by `dist` to establish differences/distances between graph GLI values. By default, absolute (“manhattan”) differences are used.
- `jitter` Should values be jittered prior to display?
- `xlab` A label for the X axis
- `ylab` A label for the Y axis
- `lm.line` Include a least-squares line?
- `...` Additional arguments to `plot`
Details

gdist.plotdiff works by taking the distances between all graphs on meas and then plotting these distances against d for all pairs of graphs (with, optionally, an added least-squares line for reference value). This can be a useful exploratory tool for relating inter-graph distances (e.g., Hamming distances) to differences on other attributes.

Value

None

Note

This function is actually quite generic, and can be used with node-level – or even non-network – data as well.

Author(s)

Carter T. Butts (butts@csci.uci.edu)

References


See Also

gdist.plotstats, gclust.boxstats, gclust.centralgraph

Examples

#Generate some random graphs with varying densities
g <- rgraph(10, 20, tprob = runif(20, 0, 1))

#Find the Hamming distances between graphs
g.h <- hdist(g)

#Plot the relationship between distance and differences in density
gdist.plotdiff(g.h, gden(g), lm.line = TRUE)

---

**gdist.plotstats**  Plot Various Graph Statistics Over a Network MDS

Description

Plots a two-dimensional metric MDS of d, with the corresponding values of meas indicated at each point. Various options are available for controlling how meas is to be displayed.

Usage

```
gdist.plotstats(d, meas, siz.lim = c(0, 0.15), rescale = "quantile", display.scale = "radius", display.type = "circleray", cex = 0.5, pch = 1, labels = NULL, pos = 1, labels.cex = 1, legend = NULL, legend.xy = NULL, legend.cex = 1, ...)```

```
Arguments

d        A matrix containing the inter-graph distances
meas     An nxm matrix containing the graph-level measures; each row must correspond
to a graph, and each column must correspond to an index
siz.lim  The minimum and maximum sizes (respectively) of the plotted symbols, given
as fractions of the total plotting range
rescale  One of “quantile” for ordinal scaling, “affine” for max-min scaling, and “nor-
        malize” for rescaling by maximum value; these determine the scaling rule to be
        used in sizing the plotting symbols
display.scale
        One of “area” or “radius”: this controls the attribute of the plotting symbol which
        is rescaled by the value of meas
display.type
        One of “circle”, “ray”, “circleray”, “poly”, or “polyray”: this determines the
        type of plotting symbol used (circles, rays, polygons, or come combination of
        these)
cex      Character expansion coefficient
pch      Point types for the base plotting symbol (not the expanded symbols which are
        used to indicate meas values)
labels   Point labels, if desired
pos      Relative position of labels (see par)
labels.cex Character expansion factor for labels
legend   Add a legend?
legend.xy x,y coordinates for legend
legend.cex Character expansion factor for legend
...      Additional arguments to plot

Details

gdist.plotstats works by performing an MDS (using cmdscale) on d, and then using the
values in meas to determine the shape of the points at each MDS coordinate. Typically, these shapes
involve rays of varying color and length indicating meas magnitude, with circles and polygons of
the appropriate radius and/or error being options as well. Various options are available (described
above) to govern the details of the data display; some tinkering may be needed in order to produce
an aesthetically pleasing visualization.

The primary use of gdist.plotstats is to explore broad relationships between graph proper-
ties and inter-graph distances. This routine complements others in the gdist and gclust family
of interstructural visualization tools.

Value

None

Note

This routine does not actually depend on the data’s being graphic in origin, and can be used with
any distance matrix/measure matrix combination.
geodist

Author(s)
Carter T. Butts ⟨buttsc@uci.edu⟩

References

See Also
gdist.plotdiff, gclust.boxstats, gclust.centralgraph

Examples
#Generate random graphs with varying density
g<-rgraph(10,20,tprob=runif(20,0,1))

#Get Hamming distances between graphs
g.h<-hdist(g)

#Plot the association of distance, density, and reciprocity
gdist.plotstats(g.h,cbind(gden(g),grecip(g)))

geodist

Fund the Numbers and Lengths of Geodesics Among Nodes in a Graph

Description
geodist uses a BFS to find the number and lengths of geodesics between all nodes of dat. Where geodesics do not exist, the value in inf.replace is substituted for the distance in question.

Usage
geodist(dat, inf.replace=Inf)

Arguments
dat an adjacency matrix.
inf.replace the value to use for geodesic distances between disconnected nodes; by default, this is equal Inf.

Details
This routine is used by a variety of other functions; many of these will allow the user to provide manually precomputed geodist output so as to prevent expensive recomputation. Note that the choice of one greater than the maximum path length for disconnected vertex pairs is non-canonical (albeit common), and some may prefer to simply treat these as missing values. geodist (without loss of generality) treats all paths as directed, a fact which should be kept in mind when interpreting geodist output.
Value

A list containing:

- `counts`: A matrix containing the number of geodesics between each pair of vertices
- `sigma`: A matrix containing the geodesic distances between each pair of vertices

Note

Before `sna` version 0.5, `inf.replace` defaulted to `NROW(dat)`.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

`component.dist`, `components`

Examples

```r
# Find geodesics on a random graph
gd <- geodist(rgraph(15))

# Examine the number of geodesics
gd$counts

# Examine the geodesic distances
gd$gdist
```

### gliop

**Return a Binary Operation on GLI Values Computed on Two Graphs**

#### Description

`gliop` is a wrapper which allows for an arbitrary binary operation on GLIs to be treated as a single call. This is particularly useful for test routines such as `cugtest` and `qaptest`.

#### Usage

```r
gliop(dat, GFUN, OP="-", g1=1, g2=2, ...)
```
gliop

Arguments

- **dat**: A graph stack
- **GFUN**: A function taking single graphs as input
- **OP**: The operator to use on the output of GFUN
- **g1**: The index of the first input graph
- **g2**: The index of the second input graph
- ...: Additional arguments to GFUN

Details

gliop operates by evaluating GFUN on the graphs indexed by g1 and g2 and returning the result of OP as applied to the GFUN output.

Value

\[ \text{OP}(\text{GFUN}(\text{dat}[g1, , ]),...),\text{GFUN}(\text{dat}[g2, , ]),...)) \]

Note

If the output of GFUN is not sufficiently well-behaved, undefined behavior may occur. Common sense is advised.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

cugtest, qaptest

Examples

```r
# Draw two random graphs
g <- rgraph(10, 2, tprob = c(0.2, 0.5))

# What is their difference in density?
gliop(g, gden, "-", 1, 2)
```
gplot produces a two-dimensional plot of graph g in stack dat. A variety of options are available to control vertex placement, display details, color, etc.

Usage

gplot(dat, g = 1, gmode = "digraph", diag = FALSE, label = c(1:dim(dat)[2]), coord = NULL, jitter = TRUE, thresh = 0, usearrows = TRUE, mode = "fruchtermanreingold", displayisolates = TRUE, interactive = FALSE, xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL, pad = 0.2, label.pad = 0.5, displaylabels = FALSE, boxed.labels = TRUE, label.pos = 0, label.bg = "white", vertex.sides = 8, vertex.rot = 0, arrowhead.cex = 1, label.cex = 1, loop.cex = 1, vertex.cex = 1, edge.col = 1, label.col = 1, vertex.col = 2, label.border = 1, vertex.border = 1, edge.lty = 1, label.lty = NULL, vertex.lty = 1, edge.lwd = 0, label.lwd = par("lwd"), edge.len = 0.5, edge.curve = 0.1, edge.steps = 50, loop.steps = 20, object.scale = 0.01, uselen = FALSE, usecurve = FALSE, suppress.axes = TRUE, vertices.last = TRUE, new = TRUE, layout.par = NULL, ...)

Arguments

dat a graph or graph stack. This data may be valued.
g integer indicating the index of the graph which is to be plotted. By default, g=1.
gmode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected; "twomode" indicates that data should be interpreted as bimodal (i.e., rows and columns are distinct vertex sets). gmode is set to "digraph" by default.
diag boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
label a vector of vertex labels, if desired; defaults to the vertex index number.
coord user-specified vertex coordinates, in an NCOL(dat)x2 matrix. Where this is specified, it will override the mode setting.
jitter boolean; should the output be jittered?
thresh real number indicating the lower threshold for tie values. Only ties of value >thresh are displayed. By default, thresh=0.
usearrows boolean; should arrows (rather than line segments) be used to indicate edges?
mode the vertex placement algorithm; this must correspond to a gplot.layout function.
displayisolates
boolean; should isolates be displayed?

interactive
boolean; should interactive adjustment of vertex placement be attempted?

xlab
x axis label.

ylab
y axis label.

xlim
the x limits (min, max) of the plot.

ylim
the y limits of the plot.

pad
amount to pad the plotting range; useful if labels are being clipped.

label.pad
amount to pad label boxes (if boxed.labels==TRUE), in character size units.

displaylabels
boolean; should vertex labels be displayed?

boxed.labels
boolean; place vertex labels within boxes?

label.pos
position at which labels should be placed, relative to vertices. 0 results in labels which are placed away from the center of the plotting region; 1, 2, 3, and 4 result in labels being placed below, to the left of, above, and to the right of vertices (respectively); and label.pos>=5 results in labels which are plotted with no offset (i.e., at the vertex positions).

label.bg
background color for label boxes (if boxed.labels==TRUE); may be a vector, if boxes are to be of different colors.

vertex.sides
number of polygon sides for vertices; may be given as a vector, if vertices are to be of different types.

vertex.rot
angle of rotation for vertices (in degrees); may be given as a vector, if vertices are to be rotated differently.

arrowhead.cex
expansion factor for edge arrowheads.

label.cex
character expansion factor for label text.

loop.cex
expansion factor for loops; may be given as a vector, if loops are to be of different sizes.

vertex.cex
expansion factor for vertices; may be given as a vector, if vertices are to be of different sizes.

edge.col
color for edges; may be given as a vector or adjacency matrix, if edges are to be of different colors.

label.col
color for vertex labels; may be given as a vector, if labels are to be of different colors.

vertex.col
color for vertices; may be given as a vector, if vertices are to be of different colors.

label.border
label border colors (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different colors.

vertex.border
border color for vertices; may be given as a vector, if vertex borders are to be of different colors.

edge.lty
line type for edge borders; may be given as a vector or adjacency matrix, if edge borders are to have different line types.

label.lty
line type for label boxes (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different line types.
vertex.lty line type for vertex borders; may be given as a vector or adjacency matrix, if vertex borders are to have different line types.

eedge.lwd line width scale for edges; if set greater than 0, edge widths are scaled by edge.lwd*dat. May be given as a vector, if label boxes are to have different line widths.

label.lwd line width for label boxes (if boxed.labels==TRUE); may be given as a vector, if label boxes are to have different line widths.

edge.len if uselen==TRUE, curved edge lengths are scaled by edge.len.

edge.curve if usecurve==TRUE, the extent of edge curvature is controlled by edge.curv. May be given as a fixed value, vector, or adjacency matrix, if edges are to have different levels of curvature.

edge.steps for curved edges (excluding loops), the number of line segments to use for the curve approximation.

loop.steps for loops, the number of line segments to use for the curve approximation.

object.scale base length for plotting objects, as a fraction of the linear scale of the plotting region. Defaults to 0.01.

uselen boolean; should we use edge.len to rescale edge lengths?

usecurve boolean; should we use edge.curve?

suppress.axes boolean; suppress plotting of axes?

vertices.last boolean; plot vertices after plotting edges?

new boolean; create a new plot? If new==FALSE, vertices and edges will be added to the existing plot.

layout.par parameters to the gplot.layout function specified in mode.

... additional arguments to plot.

Details
gplot is the standard network visualization tool within the sna library. By means of clever selection of display parameters, a fair amount of display flexibility can be obtained. Graph layout – if not specified directly using coord – is determined via one of the various available algorithms. These should be specified via the mode argument; see gplot.layout for a full list. User-supplied layout functions are also possible – see the aforementioned man page for details.

Note that where gmode=="twomode", the supplied two-mode matrix is converted to bipartite adjacency form prior to computing coordinates. If interactive==TRUE, then the user may modify the initial graph layout by selecting an individual vertex and then clicking on the location to which this vertex is to be moved; this process may be repeated until the layout is satisfactory.

Value
A two-column matrix containing the vertex positions as x,y coordinates.

Author(s)
Carter T. Butts (buttsc@uci.edu)
Alex Montgomery (ahm@stanford.edu)


References


See Also

plot, gplot.layout

Examples

gplot(rgraph(5)) # Plot a random graph
gplot(rgraph(5), usecurv=TRUE) # This time, use curved edges
gplot(rgraph(5), mode="mds") # Try an alternative layout scheme

# A colorful demonstration...
gplot(rgraph(5, diag=TRUE), diag=TRUE, vertex.cex=1:5, vertex.sides=3:8, vertex.col=1:5, vertex.border=2:6, vertex.rot=(0:4)*72, displaylabels=TRUE, label.bg="gray90")

---

**gplot.arrow**

Add Arrows or Segments to a Plot

Description

**gplot.arrow** draws a segment or arrow between two pairs of points; unlike **arrows** or **segments**, the new plot element is drawn as a polygon.

Usage

gplot.arrow(x0, y0, x1, y1, length = 0.1, angle = 20, width = 0.01, col = 1, border = 1, lty = 1, offset.head = 0, offset.tail = 0, arrowhead = TRUE, curve = 0, edge.steps = 50, ...)

Arguments

- **x0**: A vector of x coordinates for points of origin
- **y0**: A vector of y coordinates for points of origin
- **x1**: A vector of x coordinates for destination points
- **y1**: A vector of y coordinates for destination points
- **length**: Arrowhead length, in current plotting units
- **angle**: Arrowhead angle (in degrees)
- **width**: Width for arrow body, in current plotting units (can be a vector)
- **col**: Arrow body color (can be a vector)
- **border**: Arrow border color (can be a vector)
- **lty**: Arrow border line type (can be a vector)
- **offset.head**: Offset for destination point (can be a vector)
- **offset.tail**: Offset for origin point (can be a vector)
arrowhead  Boolean; should arrowheads be used? (Can be a vector)
curve  Degree of edge curvature (if any), in current plotting units (can be a vector)
edge.steps  For curved edges, the number of steps to use in approximating the curve (can be a vector)
...  Additional arguments to polygon

Details

gplot.arrow provides a useful extension of segments and arrows when fine control is needed over the resulting display. (The results also look better.) Note that edge curvature is quadratic, with curve providing the maximum horizontal deviation of the edge (left-handed). Head/tail offsets are used to adjust the end/start points of an edge, relative to the baseline coordinates; these are useful for functions like gplot, which need to draw edges incident to vertices of varying radii.

Value

None.

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

gplot, gplot.loop, polygon

Examples

#Plot two points
plot(1:2,1:2)

#Add an edge
gplot.arrow(1,1,2,2,width=0.01,col="red",border="black")

Description

Various functions which generate vertex layouts for the gplot visualization routine.

Usage

gplot.layout.adj(d, layout.par)
gplot.layout.circle(d, layout.par)
gplot.layout.circrand(d, layout.par)
gplot.layout.eigen(d, layout.par)
gplot.layout.fruchtermanreingold(d, layout.par)
gplot.layout.geodist(d, layout.par)
gplot.layout.hall(d, layout.par)
gplot.layout.kamadakawai(d, layout.par)
gplot.layout

```
gplot.layout.mds(d, layout.par)
gplot.layout.princcoord(d, layout.par)
gplot.layout.random(d, layout.par)
gplot.layout.rmds(d, layout.par)
gplot.layout.segeo(d, layout.par)
gplot.layout.seham(d, layout.par)
gplot.layout.spring(d, layout.par)
gplot.layout.springrepulse(d, layout.par)
gplot.layout.target(d, layout.par)
```

### Arguments

- **d**: an adjacency matrix, as passed by `gplot`.
- **layout.par**: a list of parameters.

### Details

Vertex layouts for network visualization pose a difficult problem – there is no single, “good” layout algorithm, and many different approaches may be valuable under different circumstances. With this in mind, `gplot` allows for the use of arbitrary vertex layout algorithms via the `gplot.layout.*` family of routines. When called, `gplot` searches for a `gplot.layout` function whose third name matches its `mode` argument (see `gplot` help for more information); this function is then used to generate the layout for the resulting plot. In addition to the routines documented here, users may add their own layout functions as needed. The requirements for a `gplot.layout` function are as follows:

1. the first argument, `d`, must be the (dichotomous) graph adjacency matrix;
2. the second argument, `layout.par`, must be a list of parameters (or `NULL`, if no parameters are specified); and
3. the return value must be a real matrix of dimension `c(2, NROW(d))`, whose rows contain the vertex coordinates.

Other than this, anything goes. (In particular, note that `layout.par` could be used to pass additional matrices, if needed.)

The `graph.layout` functions currently supplied by default are as follows:

**circle**
- This function places vertices uniformly in a circle; it takes no arguments.

**eigen**
- This function places vertices based on the eigenstructure of the adjacency matrix. It takes the following arguments:
  - `layout.par$var`: This argument controls the matrix to be used for the eigenanalysis. "symupper", "symlower", "symstrong", "symweak" invoke `symmetrize` on `d` with the respective symmetrizing rule. "user" indicates a user-supplied matrix (see below), while "raw" indicates that `d` should be used as-is. (Defaults to "raw").
  - `layout.par$evsel`: If "first", the first two eigenvectors are used; if "size", the two eigenvectors whose eigenvalues have the largest magnitude are used instead. Note that only the real portion of the associated eigenvectors is used. (Defaults to "first").
  - `layout.par$mat`: If `layout.par$var=="user"`, this matrix is used for the eigenanalysis. (No default.)

**fruchtermanreingold**
- This function generates a layout using a variant of Fruchterman and Reingold’s force-directed placement algorithm. It takes the following arguments:
  - `layout.par$niter`: This argument controls the number of iterations to be employed. (Defaults to 500.)
layout.par$max.delta  Sets the maximum change in position for any given iteration. (Defaults to NROW(d).)
layout.par$area  Sets the "area" parameter for the F-R algorithm. (Defaults to NROW(d) * 2.)
layout.par$cool.exp  Sets the cooling exponent for the annealer. (Defaults to 3.)
layout.par$repulse.rad  Determines the radius at which vertex-vertex repulsion cancels out attraction of adjacent vertices. (Defaults to area * NROW(d).)
layout.par$seed.coord  A two-column matrix of initial vertex coordinates. (Defaults to a random circular layout.)

hall  This function places vertices based on the last two eigenvectors of the Laplacian of the input matrix (Hall’s algorithm). It takes no arguments.

kamadakawai  This function generates a vertex layout using a version of the Kamada-Kawai force-directed placement algorithm. It takes the following arguments:

layout.par$niter  This argument controls the number of iterations to be employed. (Defaults to 1000.)

layout.par$sigma  Sets the base standard deviation of position change proposals. (Defaults to NROW(d) / 4.)

layout.par$init.temp  Sets the initial "temperature" for the annealing algorithm. (Defaults to 10.)

layout.par$cool.exp  Sets the cooling exponent for the annealer. (Defaults to 0.99.)

layout.par$kkconst  Sets the Kamada-Kawai vertex attraction constant. (Defaults to NROW(d) * 2.)

layout.par$elen  Provides the matrix of interpoint distances to be approximated. (Defaults to the geodesic distances of d after symmetrizing, capped at sqrt(NROW(d)).)

layout.par$seed.coord  A two-column matrix of initial vertex coordinates. (Defaults to a gaussian layout.)

mds  This function places vertices based on a metric multidimensional scaling of a specified distance matrix. It takes the following arguments:

layout.par$var  This argument controls the raw variable matrix to be used for the subsequent distance calculation and scaling. "rowcol", "row", and "col" indicate that the rows and columns (concatenated), rows, or columns (respectively) of d should be used. "rcsum" and "rcdiff" result in the sum or difference of d and its transpose being employed. "invadj" indicates that max(d) - d should be used, while "geodist" uses geodist to generate a matrix of geodesic distances from d. Alternately, an arbitrary matrix can be provided using "user". (Defaults to "rowcol").

layout.par$dist  The distance function to be calculated on the rows of the variable matrix. This must be one of the method parameters to \list{dist} ("euclidean", "maximum", "manhattan", or "canberra"), or else "none". In the latter case, no distance function is calculated, and the matrix in question must be square (with dimension dim(d)) for the routine to work properly. (Defaults to "euclidean").

layout.par$exp  The power to which distances should be raised prior to scaling. (Defaults to 2.)

layout.par$vm  If layout.par$var == "user", this matrix is used for the distance calculation. (No default.)

Note: the following layout functions are based on mds:

adj  scaling of the raw adjacency matrix, treated as similarities (using "invadj").

geodist  scaling of the matrix of geodesic distances.

rmds  euclidean scaling of the rows of d.

segeo  scaling of the squared euclidean distances between row-wise geodesic distances (i.e., approximate structural equivalence).
seham scaling of the Hamming distance between rows/columns of $d$ (i.e., another approximate structural equivalence scaling).

princoord This function places vertices based on the eigenstructure of a given correlation/covariance matrix. It takes the following arguments:

- `layout.par$var` The matrix of variables to be used for the correlation/covariance calculation. "rowcol", "col", and "row" indicate that the rows/cols, columns, or rows (respectively) of $d$ should be employed. "rcsum" "rcdiff" result in the sum or difference of $d$ and $t(d)$ being used. "user" allows for an arbitrary variable matrix to be supplied. (Defaults to "rowcol".)

- `layout.par$cor` Should the correlation matrix (rather than the covariance matrix) be used? (Defaults to TRUE.)

- `layout.par$vm` If `layout.par$var=="user"`, this matrix is used for the correlation/covariance calculation. (No default.)

random This function places vertices randomly. It takes the following argument:

- `layout.par$dist` The distribution to be used for vertex placement. Currently, the options are "unif" (for uniform distribution on the square), "uniang" (for a "gaussian donut" configuration), and "normal" (for a straight Gaussian distribution). (Defaults to "unif".)

Note: `cirtrand`, which is a frontend to the "uniang" option, is based on this function.

spring This function places vertices using a spring embedder. It takes the following arguments:

- `layout.par$mass` The vertex mass (in "quasi-kilograms"). (Defaults to 0.1.)

- `layout.par$equil` The equilibrium spring extension (in "quasi-meters"). (Defaults to 1.)

- `layout.par$k` The spring coefficient (in "quasi-Newtons per quasi-meter"). (Defaults to 0.001.)

- `layout.par$repeqdis` The point at which repulsion (if employed) balances out the spring extension force (in "quasi-meters"). (Defaults to 0.1.)

- `layout.par$kfr` The base coefficient of kinetic friction (in "quasi-Newton quasi-kilograms"). (Defaults to 0.01.)

- `layout.par$repulse` Should repulsion be used? (Defaults to FALSE.)

Note: `springrepulse` is a frontend to `spring`, with repulsion turned on.

target This function produces a "target diagram" or "bullseye" layout, using a Brandes et al.’s force-directed placement algorithm. (See also `gplot.target`.) It takes the following arguments:

- `layout.par$niter` This argument controls the number of iterations to be employed. (Defaults to 1000.)

- `layout.par$radii` This argument should be a vector of length `NROW(d)` containing vertex radii. Ideally, these should lie in the [0,1] interval (and odd behavior may otherwise result). (Defaults to the affine-transformed Freeman degree centrality scores of $d$.)

- `layout.par$minlen` Sets the minimum edge length, below which edge lengths are to be adjusted upwards. (Defaults to 0.05.)

- `layout.par$area` Sets the initial "temperature" for the annealing algorithm. (Defaults to 10.)

- `layout.par$cool.exp` Sets the cooling exponent for the annealer. (Defaults to 0.99.)

- `layout.par$maxdelta` Sets the maximum angular distance for vertex moves. (Defaults to $\pi$.)

- `layout.par$periph.outside` Boolean; should "peripheral" vertices (in the Brandes et al. sense) be placed together outside the main target area? (Defaults to FALSE.)

- `layout.par$periph.outside.offset` Radius at which to place "peripheral" vertices if `periph.outside==TRUE`. (Defaults to 1.2.)

- `layout.par$disconst` Multiplier for the Kamada-Kawai-style distance potential. (Defaults to 1.)
**layout.par$crossconst**  Multiplier for the edge crossing potential. (Defaults to 1.)

**layout.par$repconst**  Multiplier for the vertex-edge repulsion potential. (Defaults to 1.)

**layout.par$minpdis**  Sets the "minimum distance" parameter for vertex repulsion. (Defaults to 0.05.)

**Value**

A matrix whose rows contain the x,y coordinates of the vertices of d.

**Author(s)**

Carter T. Butts  ⟨buttsc@uci.edu⟩

**References**


**See Also**

`gplot`, `gplot.target`, `gplot3d.layout`, `cmdscale`, `eigen`

---

**gplot.loop**  
**Add Loops to a Plot**

**Description**

`gplot.loop` draws a "loop" at a specified location; this is used to designate self-ties in `gplot`.

**Usage**

```r
  gplot.loop(x0, y0, length = 0.1, angle = 10, width = 0.01, col = 1,
             border = 1, lty = 1, offset = 0, edge.steps = 10, radius = 1,
             arrowhead = TRUE, xctr=0, yctr=0, ...)```

**Arguments**

- `x0`: a vector of x coordinates for points of origin.
- `y0`: a vector of y coordinates for points of origin.
- `length`: arrowhead length, in current plotting units.
- `angle`: arrowhead angle (in degrees).
- `width`: width for loop body, in current plotting units (can be a vector).
- `col`: loop body color (can be a vector).
- `border`: loop border color (can be a vector).
- `lty`: loop border line type (can be a vector).
offset: offset for origin point (can be a vector).
edge.steps: number of steps to use in approximating curves.
radius: loop radius (can be a vector).
arrowhead: boolean; should arrowheads be used? (Can be a vector.)
xctr: x coordinate for the central location away from which loops should be oriented.
yctr: y coordinate for the central location away from which loops should be oriented.
...
additional arguments to polygon.

Details

gplot.loop is the companion to gplot.arrow; like the latter, plot elements produced by gplot.loop are drawn using polygon, and as such are scaled based on the current plotting device. By default, loops are drawn so as to encompass a circular region of radius radius, whose center is offset units from x0,y0 and at maximum distance from xctr,yctr. This is useful for functions like gplot, which need to draw loops incident to vertices of varying radii.

Value
None.

Author(s)
Carter T. Butts (buttsc@uci.edu)

See Also
gplot.arrow, gplot.poly

Examples

#Plot a few polygons with loops
plot(0,0,type="n",xlim=c(-2,2),ylim=c(-2,2),asp=1)
gplot.loop(c(0,0),c(1,-1),col=c(3,2),width=0.05,length=0.4,
offset=sqrt(2)/4,angle=20,radius=0.5,edge.steps=50,arrowhead=TRUE)
polygon(c(0.25,-0.25,-0.25,0.25,NA,0.25,-0.25,-0.25,0.25),
c(1.25,1.25,0.75,0.75,NA,-1.25,-1.25,-0.75,-0.75),col=c(2,3))

Description

Displays an input graph (and associated vector) as a "target diagram," with vertices restricted to lie at fixed radii from the origin. Such displays are useful ways of representing vertex characteristics and/or local structural properties for graphs of small to medium size.
Usage

\texttt{gplot.target(dat, x, circ.rad = (1:10)/10, circ.col = "blue",}
\texttt{circ.lwd = 1, circ.lty = 3, circ.lab = TRUE, circ.lab.cex = 0.75,}
\texttt{circ.lab.theta = pi, circ.lab.col = 1, circ.lab.digits = 1,}
\texttt{circ.lab.offset = 0.025, periph.outside = FALSE,}
\texttt{periph.outside.offset = 1.2, ...)}

Arguments

\begin{itemize}
  \item \texttt{dat} A graph adjacency matrix
  \item \texttt{x} A vector of vertex properties to be plotted (must match the dimensions of \texttt{dat})
  \item \texttt{circ.rad} Radii at which to draw reference circles
  \item \texttt{circ.col} Reference circle color
  \item \texttt{circ.lwd} Reference circle line width
  \item \texttt{circ.lty} Reference circle line type
  \item \texttt{circ.lab} Boolean; should circle labels be displayed?
  \item \texttt{circ.lab.cex} Expansion factor for circle labels
  \item \texttt{circ.lab.theta} Angle at which to draw circle labels
  \item \texttt{circ.lab.col} Color for circle labels
  \item \texttt{circ.lab.digits} Digits to display for circle labels
  \item \texttt{circ.lab.offset} Offset for circle labels
  \item \texttt{periph.outside} Boolean; should "peripheral" vertices be drawn together beyond the normal vertex radius?
  \item \texttt{periph.outside.offset} Radius at which "peripheral" vertices should be drawn if \texttt{periph.outside==TRUE}
  \item \texttt{...} Additional arguments to \texttt{gplot}
\end{itemize}

Details

\texttt{gplot.target} is a front-end to \texttt{gplot} which implements the target diagram layout of Brandes et al. (2003). This layout seeks to optimize various aesthetic criteria, given the constraint that all vertices lie at fixed radii from the origin (set by \texttt{x}). One important feature of this algorithm is that vertices which belong to mutual dyads (described by Brandes et al. as “core” vertices) are treated differently from vertices which do not (“peripheral” vertices). Layout is optimized for core vertices prior to placing peripheral vertices; thus, the result may be misleading if mutuality is not a salient characteristic of the data.

The layout for \texttt{gplot.target} is handled by \texttt{gplot.layout.target}; additional parameters are specified on the associated manual page. Standard arguments may be passed to \texttt{gplot}, as well.

Value

A two-column matrix of vertex positions (generated by \texttt{gplot.layout.target})

Author(s)

Carter T. Butts (buttsc@uci.edu)
gplot.vertex

References


See Also
gplot.layout.target, gplot

Examples

# Generate a random graph
g<-rgraph(15)

# Produce a target diagram, centering by betweenness
gplot.target(g,betweenness(g))

gplot.vertex  Add Vertices to a Plot

Description

_gplot.vertex_ adds one or more vertices (drawn using _polygon_) to a plot.

Usage

_gplot.vertex(x, y, radius = 1, sides = 4, border = 1, col = 2, lty = NULL, rot = 0, ...)_

Arguments

- _x_ a vector of x coordinates.
- _y_ a vector of y coordinates.
- _radius_ a vector of vertex radii.
- _sides_ a vector containing the number of sides to draw for each vertex.
- _border_ a vector of vertex border colors.
- _col_ a vector of vertex interior colors.
- _lty_ a vector of vertex border line types.
- _rot_ a vector of vertex rotation angles (in degrees).
- _..._ Additional arguments to _polygon_

Details

_gplot.vertex_ draws regular polygons of specified radius and number of sides, at the given coordinates. This is useful for routines such as _gplot_, which use such shapes to depict vertices.

Value

None
gplot3d

Three-Dimensional Visualization of Graphs

Description

gplot3d produces a three-dimensional plot of graph g in stack dat. A variety of options are available to control vertex placement, display details, color, etc.

Usage

gplot3d(dat, g = 1, gmode = "digraph", diag = FALSE, label = c(1:dim(dat)[2]), coord = NULL, jitter = TRUE, thresh = 0, mode = "fruchtermanreingold", displayisolates = TRUE, displaylabels = FALSE, xlab = NULL, ylab = NULL, zlab = NULL, vertex.radius = NULL, absolute.radius = FALSE, label.col = "gray50", edge.col = "black", vertex.col = "red", edge.alpha = 1, vertex.alpha = 1, edge.lwd = NULL, suppress.axes = TRUE, new = TRUE, bg.col = "white", layout.par = NULL)

Arguments

dat a graph or graph stack. This data may be valued.
g integer indicating the index of the graph (from dat) which is to be displayed.
gmode string indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected; "twomode" indicates that data should be interpreted as two-mode (i.e., rows and columns are distinct vertex sets).
diag boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops.
label a vector of vertex labels; setting this to a zero-length string (e.g., "") omits
coord user-specified vertex coordinates, in an NCOL(dat) x 3 matrix. Where this is specified, it will override the mode setting.
jitter boolean; should vertex positions be jittered?
thresh real number indicating the lower threshold for tie values. Only ties of value >thresh are displayed.

mode the vertex placement algorithm; this must correspond to a gplot3d.layout function.

displayisolates boolean; should isolates be displayed?

displaylabels boolean; should vertex labels be displayed?

xlab X axis label.

ylab Y axis label.

zlab Z axis label.

vertex.radius vertex radius, relative to the baseline (which is set based on layout features); may be given as a vector, if radii vary across vertices.

absolute.radius vertex radius, specified in absolute terms; this may be given as a vector.

label.col color for vertex labels; may be given as a vector, if labels are to be of different colors.

displaylabels boolean; should vertex labels be displayed?

displaylabels boolean; should vertex labels be displayed?

edge.col color for edges; may be given as a vector or adjacency matrix, if edges are to be of different colors.

vertex.col color for vertices; may be given as a vector, if vertices are to be of different colors.

vertex.col color for vertices; may be given as a vector, if vertices are to be of different colors.

edge.alpha alpha (transparency) values for edges; may be given as a vector or adjacency matrix, if edge transparency is to vary.

vertex.alpha alpha (transparency) values for vertices; may be given as a vector, if vertex transparency is to vary.

displaylabels boolean; should vertex labels be displayed?

displaylabels boolean; should vertex labels be displayed?

edge.lwd line width scale for edges; if set greater than 0, edge widths are rescaled by edge.lwd*dat. May be given as a vector or adjacency matrix, if edges are to have different line widths.

suppress.axes boolean; suppress plotting of axes?

new boolean; create a new plot? If new==FALSE, the RGL device will not be cleared prior to adding vertices and edges.

bg.col background color for display.

layout.par list of parameters to the gplot.layout function specified in mode.

Details
gplot3d is the three-dimensional companion to gplot. As with the latter, clever manipulation of parameters can allow for a great deal of flexibility in the resulting display. (Displays produced by gplot3d are also interactive, to the extent supported by rgl.) If vertex positions are not specified directly using coord, vertex layout is determined via one of the various available algorithms. These should be specified via the mode argument; see gplot3d.layout for a full list. User-supplied layout functions are also possible - see the aforementioned man page for details.

Note that where gmode="twomode", the supplied two-mode matrix is converted to bipartite adjacency form prior to computing coordinates. It may be desirable to use parameters such as vertex.col to differentiate row and column vertices.
Value
A three-column matrix containing vertex coordinates

Requires
rgl

Author(s)
Carter T. Butts ⟨buttsc@uci.edu⟩

References

See Also
gplot, gplot3d.layout, rgl

Examples

```r
## Not run:
# A three-dimensional grid...
gplot3d(rgws(1,5,3,1,0))

# ...rewired...
gplot3d(rgws(1,5,3,1,0.05))

# ...some more!
gplot3d(rgws(1,5,3,1,0.2))
## End(Not run)
```

---

**gplot3d.arrow**

Add Arrows a Three-Dimensional Plot

Description
gplot3d.arrow draws an arrow between two pairs of points.

Usage
gplot3d.arrow(a, b, radius, color = "white", alpha = 1)

Arguments

- **a**: a vector or three-column matrix containing origin X,Y,Z coordinates.
- **b**: a vector or three-column matrix containing origin X,Y,Z coordinates.
- **radius**: the arrow radius, in current plotting units. May be a vector, if multiple arrows are to be drawn.
- **color**: the arrow color. May be a vector, if multiple arrows are being drawn.
- **alpha**: alpha (transparency) value(s) for arrows. (May be a vector.)
Details

gplot3d.arrow draws one or more three-dimensional "arrows" from the points given in a to those given in b. Note that the "arrows" are really cones, narrowing in the direction of the destination point.

Value

None.

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

gplot3d, gplot3d.loop, rgl.primitive

gplot3d.layout
Vertex Layout Functions for gplot3d

Description

Various functions which generate vertex layouts for the gplot3d visualization routine.

Usage

gplot3d.layout.adj(d, layout.par)
gplot3d.layout.eigen(d, layout.par)
gplot3d.layout.fruchtermanreingold(d, layout.par)
gplot3d.layout.geodist(d, layout.par)
gplot3d.layout.hall(d, layout.par)
gplot3d.layout.kamadakawai(d, layout.par)
gplot3d.layout.mds(d, layout.par)
gplot3d.layout.princoord(d, layout.par)
gplot3d.layout.random(d, layout.par)
gplot3d.layout.rmds(d, layout.par)
gplot3d.layout.segeo(d, layout.par)
gplot3d.layout.seham(d, layout.par)

Arguments

d an adjacency matrix, as passed by gplot3d.
layout.par a list of parameters.

Details

Like gplot, gplot3d allows for the use of arbitrary vertex layout algorithms via the gplot3d.layout.* family of routines. When called, gplot3d searches for a gplot3d.layout function whose third name matches its mode argument (see gplot3d help for more information); this function is then used to generate the layout for the resulting plot. In addition to the routines documented here, users may add their own layout functions as needed. The requirements for a gplot3d.layout function are as follows:
1. the first argument, \(d\), must be the (dichotomous) graph adjacency matrix;
2. the second argument, `layout.par`, must be a list of parameters (or NULL, if no parameters are specified); and
3. the return value must be a real matrix of dimension \(c(3, \text{NROW}(d))\), whose rows contain the vertex coordinates.

Other than this, anything goes. (In particular, note that `layout.par` could be used to pass additional matrices, if needed.)

The `gplot3d.layout` functions currently supplied by default are as follows:

**eigen** This function places vertices based on the eigenstructure of the adjacency matrix. It takes the following arguments:

- `layout.par$var` This argument controls the matrix to be used for the eigenanalysis. "symupper", "symlower", "symstrong", "symweak" invoke `symmetrize` on \(d\) with the respective symmetrizing rule. "user" indicates a user-supplied matrix (see below), while "raw" indicates that \(d\) should be used as-is. (Defaults to "raw".)
- `layout.par$evsel` If "first", the first three eigenvectors are used; if "size", the three eigenvectors whose eigenvalues have the largest magnitude are used instead. Note that only the real portion of the associated eigenvectors is used. (Defaults to "first".)
- `layout.par$mat` If `layout.par$var="user"`, this matrix is used for the eigenanalysis. (No default.)

**fruchtermanreingold** This function generates a layout using a variant of Fruchterman and Reingold’s force-directed placement algorithm. It takes the following arguments:

- `layout.par$niter` This argument controls the number of iterations to be employed. (Defaults to 300.)
- `layout.par$max.delta` Sets the maximum change in position for any given iteration. (Defaults to \(\text{NROW}(d)\).)
- `layout.par$volume` Sets the "volume" parameter for the F-R algorithm. (Defaults to \(\text{NROW}(d)^3\).)
- `layout.par$cool.exp` Sets the cooling exponent for the annealer. (Defaults to 3.)
- `layout.par$repulse.rad` Determines the radius at which vertex-vertex repulsion cancels out attraction of adjacent vertices. (Defaults to \(\text{volume} \times \text{NROW}(d)\).)
- `layout.par$seed.coord` A three-column matrix of initial vertex coordinates. (Defaults to a random spherical layout.)

**hall** This function places vertices based on the last three eigenvectors of the Laplacian of the input matrix (Hall’s algorithm). It takes no arguments.

**kamadakawai** This function generates a vertex layout using a version of the Kamada-Kawai force-directed placement algorithm. It takes the following arguments:

- `layout.par$niter` This argument controls the number of iterations to be employed. (Defaults to 1000.)
- `layout.par$sigma` Sets the base standard deviation of position change proposals. (Defaults to \(\text{NROW}(d)/4\).)
- `layout.par$init.temp` Sets the initial "temperature" for the annealing algorithm. (Defaults to 10.)
- `layout.par$cool.exp` Sets the cooling exponent for the annealer. (Defaults to 0.99.)
- `layout.par$kconst` Sets the Kamada-Kawai vertex attraction constant. (Defaults to \(\text{NROW}(d)^3\).)
- `layout.par$elen` Provides the matrix of interpoint distances to be approximated. (Defaults to the geodesic distances of \(d\) after symmetrizing, capped at \(\sqrt{\text{NROW}(d)}\).)
- `layout.par$seedcoord` A three-column matrix of initial vertex coordinates. (Defaults to a gaussian layout.)
gplot3d.layout

mds This function places vertices based on a metric multidimensional scaling of a specified distance matrix. It takes the following arguments:

layout.par$var This argument controls the raw variable matrix to be used for the subsequent distance calculation and scaling. "rowcol", "row", and "col" indicate that the rows and columns (concatenated), rows, or columns (respectively) of d should be used. "rcsum" and "rcdiff" result in the sum or difference of d and its transpose being employed. "inva" indicates that max(d)−d should be used, while "gdist" uses gdist to generate a matrix of geodesic distances from d. Alternately, an arbitrary matrix can be provided using "user". (Defaults to "rowcol").

layout.par$dist The distance function to be calculated on the rows of the variable matrix. This must be one of the method parameters to \list{dist}("euclidean", "maximum", "manhattan", or "canberra"), or else "none". In the latter case, no distance function is calculated, and the matrix in question must be square (with dimension \text{dim}(d)) for the routine to work properly. (Defaults to "euclidean").

layout.par$exp The power to which distances should be raised prior to scaling. (Defaults to 2.)

layout.par$vm If layout.par$var=="user", this matrix is used for the distance calculation. (No default.)

Note: the following layout functions are based on mds:

adj scaling of the raw adjacency matrix, treated as similarities (using "inva").

gdist scaling of the matrix of geodesic distances.

rmds euclidean scaling of the rows of d.

segeo scaling of the squared euclidean distances between row-wise geodesic distances (i.e., approximate structural equivalence).

seham scaling of the Hamming distance between rows/columns of d (i.e., another approximate structural equivalence scaling).

princoord This function places vertices based on the eigenstructure of a given correlation/covariance matrix. It takes the following arguments:

layout.par$var The matrix of variables to be used for the correlation/covariance calculation. "rowcol", "col", and "row" indicate that the rows/cols, columns, or rows (respectively) of d should be employed. "rcsum" "rcdiff" result in the sum or difference of d and t(d) being used. "user" allows for an arbitrary variable matrix to be supplied. (Defaults to "rowcol").

layout.par$cor Should the correlation matrix (rather than the covariance matrix) be used? (Defaults to TRUE.)

layout.par$vm If layout.par$var=="user", this matrix is used for the correlation/covariance calculation. (No default.)

random This function places vertices randomly. It takes the following argument:

layout.par$dist The distribution to be used for vertex placement. Currently, the options are "unif" (for uniform distribution on the unit cube), "uniang" (for a "gaussian sphere" configuration), and "normal" (for a straight Gaussian distribution). (Defaults to "unif").

Value

A matrix whose rows contain the x,y,z coordinates of the vertices of d.

Author(s)

Carter T. Butts (butts@uci.edu)
gplot3d.loop

Add Loops to a Three-Dimensional Plot

Description

gplot3d.loop draws a "loop" at a specified location; this is used to designate self-ties in gplot3d.

Usage

gplot3d.loop(a, radius, color = "white", alpha = 1)

Arguments

- **a**: a vector or three-column matrix containing origin X,Y,Z coordinates.
- **radius**: the loop radius, in current plotting units. May be a vector, if multiple loops are to be drawn.
- **color**: the loop color. May be a vector, if multiple loops are being drawn.
- **alpha**: alpha (transparency) value(s) for loops. (May be a vector.)

Details

gplot3d.loop is the companion to gplot3d.arrow. The "loops" produced by this routine currently look less like loops than like "hats" – they are noticeable as spike-like structures which protrude from vertices. Eventually, something more attractive will be produced by this routine.

Value

None.

Author(s)

Carter T. Butts (buttc@uci.edu)

See Also

gplot3d.arrow, gplot3d.rgl

References


**graphcent**

*Compute the (Harary) Graph Centrality Scores of Network Positions*

**Description**

`graphcent` takes a graph stack (`dat`) and returns the Harary graph centralities of positions within one graph (indicated by `nodes` and `g`, respectively). Depending on the specified mode, graph centrality on directed or undirected geodesics will be returned; this function is compatible with `centralization`, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by `centralization` to normalize the observed centralization score).

**Usage**

```r
graphcent(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, tmaxdev=FALSE, cmode="directed", geodist.precomp=NULL, rescale=FALSE)
```

**Arguments**

- `dat` : Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
- `g` : Integer indicating the index of the graph for which centralities are to be calculated. By default, `g==1`.
- `nodes` : List indicating which nodes are to be included in the calculation. By default, all nodes are included.
- `gmode` : String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `gmode` is set to "digraph" by default.
- `diag` : Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.
- `tmaxdev` : Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, `tmaxdev==FALSE`.
- `cmode` : String indicating the type of graph centrality being computed (directed or undirected geodesics).
- `geodist.precomp` : A `geodist` object precomputed for the graph to be analyzed (optional)
- `rescale` : If true, centrality scores are rescaled such that they sum to 1.

**Details**

The Harary graph centrality of a vertex `v` is equal to \( \frac{1}{\max_u d(v,u)} \), where `d(v,u)` is the geodesic distance from `v` to `u`. Vertices with low graph centrality scores are likely to be near the “edge” of a graph, while those with high scores are likely to be near the “middle.” Compare this with `closeness`, which is based on the reciprocal of the sum of distances to all other vertices (rather than simply the maximum).
Value

A vector containing the centrality scores

Note

Judicious use of `geodist.precomp` can save a great deal of time when computing multiple path-based indices on the same network.

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

References


See Also

centralization

Examples

```r
# Draw a random graph with 10 members
g <- rgraph(10)
greducip(g) # Compute centrality scores
```

---

**grecip**

*Compute the Reciprocity of an Input Graph or Graph Stack*

**Description**

`grecip` calculates the dyadic reciprocity of the elements of `dat` selected by `g`.

**Usage**

```r
grecip(dat, g = NULL, measure = c("dyadic", "edgewise"))
```

**Arguments**

- `dat`: An adjacency matrix, or stack thereof
- `g`: A vector indicating which matrices to evaluate (optional)
- `measure`: One of "dyadic" (default) or "edgewise"

**Details**

The dyadic reciprocity of a graph is the proportion of dyads which are symmetric; this is computed and returned by `grecip` for the graphs indicated. Note that the dyadic reciprocity is distinct from the *edgewise* or *tie reciprocity*, which is the proportion of *edges* which are reciprocated. This latter form may be obtained by setting `measure="edgewise"`. 
Value

The graph reciprocity value(s)

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

mutuality, symmetrize

Examples

# Calculate the dyadic reciprocity scores for some random graphs
grecip(rgraph(10,5))

Description

gscor finds the product-moment structural correlation between the adjacency matrices of graphs indicated by \( g_1 \) and \( g_2 \) in stack \( \text{dat} \) (or possibly \( \text{dat2} \)) given exchangeability list \( \text{exchange.list} \). Missing values are permitted.

Usage

gscor(dat, dat2=NULL, g1=c(1:dim(dat)[1]), g2=c(1:dim(dat)[1]),
      diag=FALSE, mode="digraph", method="anneal", reps=1000,
      prob.init=0.9, prob.decay=0.85, freeze.time=25,
      full.neighborhood=TRUE, exchange.list=rep(0, dim(dat)[2]))

Arguments

dat A graph stack
dat2 Optionally, a second graph stack
g1 The indices of \( \text{dat} \) reflecting the first set of graphs to be compared; by default, all members of \( \text{dat} \) are included
g2 The indices or \( \text{dat} \) (or \( \text{dat2} \), if applicable) reflecting the second set of graphs to be compared; by default, all members of \( \text{dat} \) are included.
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.
### gscor

**Method**  
Method to be used to search the space of accessible permutations; must be one of “none”, “exhaustive”, “anneal”, “hillclimb”, or “mc”.

**reps**  
Number of iterations for monte carlo method.

**prob.init**  
Initial acceptance probability for the annealing routine.

**prob.decay**  
Cooling multiplier for the annealing routine.

**freeze.time**  
Freeze time for the annealing routine.

**full.neighborhood**  
Should the annealer evaluate the full neighborhood of pair exchanges at each iteration?

**exchange.list**  
Information on which vertices are exchangeable (see below); this must be a single number, a vector of length n, or a nx2 matrix.

### Details

The structural correlation coefficient between two graphs $G$ and $H$ is defined as

$$
\text{scor}(G, H | L_G, L_H) = \max_{L_G, L_H} \text{cor}(\ell(G), \ell(H))
$$

where $L_G$ is the set of accessible permutations/labelings of $G$, $\ell(G)$ is a permutation/relabeling of $G$, and $\ell(G) \in L_G$. The set of accessible permutations on a given graph is determined by the theoretical exchangeability of its vertices; in a nutshell, two vertices are considered to be theoretically exchangeable for a given problem if all predictions under the conditioning theory are invariant to a relabeling of the vertices in question (see Butts and Carley (2001) for a more formal exposition). Where no vertices are exchangeable, the structural correlation becomes the simple graph correlation. Where all vertices are exchangeable, the structural correlation reflects the correlation between unlabeled graphs; other cases correspond to correlation under partial labeling.

The accessible permutation set is determined by the `exchange.list` argument, which is dealt with in the following manner. First, `exchange.list` is expanded to fill an nx2 matrix. If `exchange.list` is a single number, this is trivially accomplished by replication; if `exchange.list` is a vector of length n, the matrix is formed by cbinding two copies together. If `exchange.list` is already an nx2 matrix, it is left as-is. Once the nx2 exchangeability matrix has been formed, it is interpreted as follows: columns refer to graphs 1 and 2, respectively; rows refer to their corresponding vertices in the original adjacency matrices; and vertices are taken to be theoretically exchangeable iff their corresponding exchangeability matrix values are identical. To obtain an unlabeled graph correlation (the default), then, one could simply let `exchange.list` equal any single number. To obtain the standard graph correlation, one would use the vector 1:n.

Because the set of accessible permutations is, in general, very large ($o(n!)$), searching the set for the maximum correlation is a non-trivial affair. Currently supported methods for estimating the structural correlation are hill climbing, simulated annealing, blind monte carlo search, or exhaustive search (it is also possible to turn off searching entirely). Exhaustive search is not recommended for graphs larger than size 8 or so, and even this may take days; still, this is a valid alternative for small graphs. Blind monte carlo search and hill climbing tend to be suboptimal for this problem and are not, in general recommended, but they are available if desired. The preferred (and default) option for permutation search is simulated annealing, which seems to work well on this problem (though some tinkering with the annealing parameters may be needed in order to get optimal performance). See the help for `lab.optimize` for more information regarding these options.

Structural correlation matrices are p.s.d., and are p.d. so long as no graph within the set is a linear combination of any other under any accessible permutation. Their eigendecompositions are meaningful and they may be used in linear subspace analyses, so long as the researcher is careful...
to interpret the results in terms of the appropriate set of accessible labelings. Classical null hypoth-
thesis tests should not be employed with structural correlations, and QAP tests are almost never
appropriate (save in the uniquely labeled case). See \texttt{cugtest} for a more reasonable alternative.

\textbf{Value}

An estimate of the structural correlation matrix

\textbf{Warning}

The search process can be \textit{very slow}, particularly for large graphs. In particular, the \textit{exhaustive}
method is order factorial, and will take approximately forever for unlabeled graphs of size greater
than about 7-9.

\textbf{Note}

Consult Butts and Carley (2001) for advice and examples on theoretical exchangeability.

\textbf{Author(s)}

Carter T. Butts (buttsc@uci.edu)

\textbf{References}


\textbf{See Also}

\texttt{gscov, gcor, gcov}

\textbf{Examples}

\begin{verbatim}
#Generate two random graphs
g.1<-rgraph(5)
g.2<-rgraph(5)

#Copy one of the graphs and permute it
perm<-sample(1:5)
g.3<-g.2[perm,perm]

#What are the structural correlations between the labeled graphs?
gscor(g.1,g.2,exchange.list=1:5)
gscor(g.1,g.3,exchange.list=1:5)
gscor(g.2,g.3,exchange.list=1:5)

#What are the structural correlations between the underlying
#unlabeled graphs?
gscor(g.1,g.2)
gscor(g.1,g.3)
gscor(g.2,g.3)
\end{verbatim}
gscov

Find the Structural Covariance(s) Between Two or More Graphs

Description

**gscov** finds the structural covariance between the adjacency matrices of graphs indicated by `g1` and `g2` in stack `dat` (or possibly `dat2`) given exchangeability list `exchange.list`. Missing values are permitted.

Usage

```r
gscov(dat, dat2=NULL, g1=c(1:dim(dat)[1]), g2=c(1:dim(dat)[1]),
      diag=FALSE, mode="digraph", method="anneal", reps=1000,
      prob.init=0.9, prob.decay=0.85, freeze.time=25,
      full.neighborhood=TRUE, exchange.list=rep(0, dim(dat)[2]))
```

Arguments

- `dat` A graph stack
- `dat2` Optionally, a second graph stack
- `g1` The indices of `dat` reflecting the first set of graphs to be compared; by default, all members of `dat` are included
- `g2` The indices or `dat` (or `dat2`, if applicable) reflecting the second set of graphs to be compared; by default, all members of `dat` are included.
- `diag` Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.
- `mode` String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `mode` is set to "digraph" by default.
- `method` Method to be used to search the space of accessible permutations; must be one of "none", "exhaustive", "anneal", "hillclimb", or "mc".
- `reps` Number of iterations for monte carlo method.
- `prob.init` Initial acceptance probability for the annealing routine.
- `prob.decay` Cooling multiplier for the annealing routine.
- `freeze.time` Freeze time for the annealing routine.
- `full.neighborhood` Should the annealer evaluate the full neighborhood of pair exchanges at each iteration?
- `exchange.list` Information on which vertices are exchangeable (see below); this must be a single number, a vector of length n, or a nx2 matrix.

Details

The structural covariance between two graphs G and H is defined as

\[
scov(G, H | L_G, L_H) = \max_{L_G, L_H} \text{cov}(\ell(G), \ell(H))
\]
where $L_G$ is the set of accessible permutations/labelings of $G$, $\ell(G)$ is a permutation/labeling of $G$, and $\ell(G) \in L_G$. The set of accessible permutations on a given graph is determined by the theoretical exchangeability of its vertices; in a nutshell, two vertices are considered to be theoretically exchangeable for a given problem if all predictions under the conditioning theory are invariant to a relabeling of the vertices in question (see Butts and Carley (2001) for a more formal exposition). Where no vertices are exchangeable, the structural covariance becomes the simple graph covariance. Where all vertices are exchangeable, the structural covariance reflects the covariance between unlabeled graphs; other cases correspond to covariance under partial labeling.

The accessible permutation set is determined by the exchange.list argument, which is dealt with in the following manner. First, exchange.list is expanded to fill an nx2 matrix. If exchange.list is a single number, this is trivially accomplished by replication; if exchange.list is a vector of length n, the matrix is formed by cbinding two copies together. If exchange.list is already an nx2 matrix, it is left as-is. Once the nx2 exchangeability matrix has been formed, it is interpreted as follows: columns refer to graphs 1 and 2, respectively; rows refer to their corresponding vertices in the original adjacency matrices; and vertices are taken to be theoretically exchangeable iff their corresponding exchangeability matrix values are identical. To obtain an unlabeled graph covariance (the default), then, one could simply let exchange.list equal any single number. To obtain the standard graph covariance, one would use the vector 1:n.

Because the set of accessible permutations is, in general, very large ($o(n!)$), searching the set for the maximum covariance is a non-trivial affair. Currently supported methods for estimating the structural covariance are hill climbing, simulated annealing, blind monte carlo search, or exhaustive search (it is also possible to turn off searching entirely). Exhaustive search is not recommended for graphs larger than size 8 or so, and even this may take days; still, this is a valid alternative for small graphs. Blind monte carlo search and hill climbing tend to be suboptimal for this problem and are not, in general recommended, but they are available if desired. The preferred (and default) option for permutation search is simulated annealing, which seems to work well on this problem (though some tinkering with the annealing parameters may be needed in order to get optimal performance). See the help for lab.optimize for more information regarding these options.

Structural covariance matrices are p.s.d., and are p.d. so long as no graph within the set is a linear combination of any other under any accessible permutation. Their eigendecompositions are meaningful and they may be used in linear subspace analyses, so long as the researcher is careful to interpret the results in terms of the appropriate set of accessible labelings. Classical null hypothesis tests should not be employed with structural covariances, and QAP tests are almost never appropriate (save in the uniquely labeled case). See cugtest for a more reasonable alternative.

**Value**

An estimate of the structural covariance matrix

**Warning**

The search process can be very slow, particularly for large graphs. In particular, the exhaustive method is order factorial, and will take approximately forever for unlabeled graphs of size greater than about 7-9.

**Note**

Consult Butts and Carley (2001) for advice and examples on theoretical exchangeability.

**Author(s)**

Carter T. Butts (buttsc@uci.edu)
gtrans

Compute the Transitivity of an Input Graph or Graph Stack

Description

gtrans returns the transitivity of the elements of dat selected by g, using the definition of measure. Triads involving missing values are omitted from the analysis.

Usage

gtrans(dat, g=NULL, diag=FALSE, mode="digraph", measure = c("weak", "strong", "weakcensus", "strongcensus"))

Arguments

dat A graph or graph stack

g A vector indicating the graphs which are to be analyzed; by default, all graphs are analyzed

diag A boolean indicating whether or not diagonal entries (loops) are to be taken as valid data

mode "digraph" if directed triads are sought, or else "graph"

measure One of "weak" (default), "strong", "weakcensus", or "strongcensus"

References


See Also

gscor, gcov, gcor

Examples

#Generate two random graphs
g.1<-rgraph(5)
g.2<-rgraph(5)

#Copy one of the graphs and permute it
perm<-sample(1:5)
g.3<-g.2[perm,perm]

#What are the structural covariances between the labeled graphs?
gscov(g.1,g.2,exchange.list=1:5)
gscov(g.1,g.3,exchange.list=1:5)
gscov(g.2,g.3,exchange.list=1:5)

#What are the structural covariances between the underlying unlabeled graphs?
gscov(g.1,g.2)
gscov(g.1,g.3)
gscov(g.2,g.3)
Details

Transitivity is a triadic, algebraic structural constraint. In its weak form, the transitive constraint corresponds to \( a \rightarrow b \rightarrow c \Rightarrow a \rightarrow c \). In the corresponding strong form, the constraint is \( a \rightarrow b \rightarrow c \Leftrightarrow a \rightarrow c \). (Note that the weak form is that most commonly employed.) Where \texttt{measure=="weak"}, the fraction of potentially intransitive triads obeying the weak condition is returned. With the \texttt{measure=="weaccensus"} setting, by contrast, the total number of transitive triads is computed. The strong versions of the measures are similar to the above, save in that the set of all triads is considered (since all are “at risk” for intransitivity).

Value

A vector of transitivity scores

Note

In version 0.3, the strong form was the default (and only) option for \texttt{gtrans}.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

\texttt{triad.classify, cugtest}

Examples

```r
#Draw some random graphs
g <- rgraph(5,10)

#Find transitivity scores
gtrans(g)
```

---

\texttt{gvectorize} \hspace{1cm} \textit{Vectorization of Adjacency Matrices}

Description

\texttt{gvectorize} takes an input graph stack and converts it into a corresponding number of vectors by row concatenation.

Usage

\texttt{gvectorize(mats, mode="digraph", diag=FALSE, censor.as.na=TRUE)}
Arguments

mats
One or more adjacency matrices (in a graph stack)

mode
“digraph” if data is taken to be directed, else “graph”

diag
Boolean indicating whether diagonal entries (loops) are taken to contain meaningful data

censor.as.na
If TRUE, code unused parts of the adjacency matrix as NAs prior to vectorizing; otherwise, unused parts are simply removed

Details

The output of gvectorize is a matrix in which each column corresponds to an input graph, and each row corresponds to an edge. The columns of the output matrix are formed by simple row-concatenation of the original adjacency matrices, possibly after removing cells which are not meaningful (if censor.as.na==FALSE). This is useful when preprocessing edge sets for use with glm or the like.

Value

An nxk matrix, where n is the number of arcs and k is the number of graphs; if censor.as.na==FALSE, n will be reflect the relevant number of uncensored arcs.

Author(s)

Carter T. Butts (buttsc@uci.edu)

Examples

#Draw two random graphs
g<-rgraph(10,2)

#Examine the vectorized form of the adjacency structure
gvectorize(g)

hdist  Find the Hamming Distances Between Two or More Graphs

Description

hdist returns the Hamming distance between the labeled graphs g1 and g2 in stack dat for dichotomous data, or else the absolute (manhattan) distance. If normalize is true, this distance is divided by its dichotomous theoretical maximum (conditional on |V(G)|).

Usage

hdist(dat, dat2=NULL, g1=c(1:dim(dat)[1]), g2=c(1:dim(dat)[1]), normalize=FALSE, diag=FALSE, mode="digraph")
**Arguments**

- **dat**: Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. This data need not be dichotomous, and missing values are allowed.
- **dat2**: A second data array (optional)
- **g1**: A vector indicating which graphs to compare (by default, all elements of `dat`)
- **g2**: A vector indicating against which the graphs of `g1` should be compared (by default, all graphs)
- **normalize**: Divide by the number of available dyads?
- **diag**: Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.
- **mode**: String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `mode` is set to "digraph" by default.

**Details**

The Hamming distance between two labeled graphs $G_1$ and $G_2$ is equal to $|\{e : (e \in E(G_1), e \not\in E(G_2)) \wedge (e \not\in E(G_1), e \in E(G_2))\}|$. In more prosaic terms, this may be thought of as the number of addition/deletion operations required to turn the edge set of $G_1$ into that of $G_2$. The Hamming distance is a highly general measure of structural similarity, and forms a metric on the space of graphs (simple or directed). Users should be reminded, however, that the Hamming distance is extremely sensitive to nodal labeling, and should not be employed directly when nodes are interchangeable. The structural distance (Butts and Carley (2001)), implemented in `structdist`, provides a natural generalization of the Hamming distance to the more general case of unlabeled graphs.

Null hypothesis testing for Hamming distances is available via `cugtest`, and `qaptest`; graphs which minimize the Hamming distances to all members of a graph set can be found by `centralgraph`. For an alternative means of comparing the similarity of graphs, consider `gcor`.

**Value**

A matrix of Hamming distances

**Note**

For non-dichotomous data, the distance which is returned is simply the sum of the absolute edge-wise differences.

**Author(s)**

Carter T. Butts (butsc@uci.edu)

**References**


hierarchy

Compute Graph Hierarchy Scores

Description

hierarchy takes a graph stack (dat) and returns reciprocity or Krackhardt hierarchy scores for the graphs selected by g.

Usage

hierarchy(dat, g=1:stackcount(dat), measure=c("reciprocity", "krackhardt"))

Arguments

dat A graph or graph stack
g Index values for the graphs to be utilized; by default, all graphs are selected
measure One of "reciprocity" or "krackhardt"

Details

Hierarchy measures quantify the extent of asymmetry in a structure; the greater the extent of asymmetry, the more hierarchical the structure is said to be. (This should not be confused with how centralized the structure is, i.e., the extent to which centralities of vertex positions are highly concentrated.) hierarchy provides two measures (selected by the measure argument) as follows:

1. reciprocity: This setting returns one minus the dyadic reciprocity for each input graph (see grecip)
2. krackhardt: This setting returns the Krackhardt hierarchy score for each input graph. The Krackhardt hierarchy is defined as the fraction of non-null dyads in the reachability graph which are asymmetric. Thus, when no directed paths are reciprocated (e.g., in an in/outtree), Krackhardt hierarchy is equal to 1; when all such paths are reciprocated, by contrast (e.g., in a cycle or clique), the measure falls to 0.

Hierarchy is one of four measures (connectedness, efficiency, hierarchy, and lubness) suggested by Krackhardt for summarizing hierarchical structures. Each corresponds to one of four axioms which are necessary and sufficient for the structure in question to be an outtree; thus, the measures will be equal to 1 for a given graph iff that graph is an outtree. Deviations from unity can be interpreted in terms of failure to satisfy one or more of the outtree conditions, information which may be useful in classifying its structural properties.

See Also

sdmat, structdist

Examples

# Get some random graphs
g<-rgraph(5,5,tprob=runif(5,0,1))

# Find the Hamming distances
hdist(g)
Note that hierarchy is inherently density-constrained: as densities climb above 0.5, the proportion of mutual dyads must (by the pigeonhole principle) increase rapidly, thereby reducing possibilities for asymmetry. Thus, the interpretation of hierarchy scores should take density into account, particularly if density is artifactual (e.g., due to a particular dichotomization procedure).

Value

A vector of hierarchy scores

Note

The four Krackhardt indices are, in general, nondegenerate for a relatively narrow band of size/density combinations (efficiency being the sole exception). This is primarily due to their dependence on the reachability graph, which tends to become complete rapidly as size/density increase. See Krackhardt (1994) for a useful simulation study.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

c connectedness, efficiency, hierarchy, lubness, grecip, mutuality, dyad.census

Examples

# Get hierarchy scores for graphs of varying densities
hierarchy(rgraph(10, 5, tprob = c(0.1, 0.25, 0.5, 0.75, 0.9)), measure = "reciprocity")
hierarchy(rgraph(10, 5, tprob = c(0.1, 0.25, 0.5, 0.75, 0.9)), measure = "krackhardt")

infocent Find Information Centrality Scores of Network Positions

Description

infocent takes a graph stack (dat) and returns the information centralities of positions within one graph (indicated by nodes and g, respectively). This function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).
Usage

infocent(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph",
        diag=FALSE, cmode="weak", tmaxdev=FALSE, rescale=FALSE, tol=1e-20)

Arguments

dat     Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
g     Integer indicating the index of the graph for which centralities are to be calculated. By default, g==1.
nodes     List indicating which nodes are to be included in the calculation. By default, all nodes are included.
gmode     String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. This is currently ignored.
diag     Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
cmode     The rule to be used by symmetrize when symmetrizing dichotomous data; must be one of "weak" (for an OR rule), "strong" for an AND rule), "upper" (for a max rule), or "lower" (for a min rule). Set to "weak" by default, this parameter obviously has no effect on symmetric data.
tmaxdev     Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.
rescale     If true, centrality scores are rescaled such that they sum to 1.
tol     Tolerance for near-singularities during matrix inversion (see solve)

Details

Actor information centrality is a hybrid measure which relates to both path-length indices (e.g., closeness, graph centrality) and to walk-based eigenmeasures (e.g., eigenvector centrality, Bonacich power). In particular, the information centrality of a given actor can be understood to be the harmonic average of the “bandwidth” for all paths originating with said individual (where the bandwidth is taken to be inversely related to path length). Formally, the index is constructed as follows. First, we take \( G \) to be an undirected (but possibly valued) graph – symmetrizing if necessary – with (possibly valued) adjacency matrix \( A \). From this, we remove all isolates (whose information centralities are zero in any event) and proceed to create the weighted connection matrix

\[
C = B^{-1}
\]

where \( B \) is a pseudo-adjacency matrix formed by replacing the diagonal of \( 1 - A \) with one plus each actor’s degree. Given the above, let \( T \) be the trace of \( C \) with sum \( S_T \), and let \( S_R \) be an arbitrary row sum (all rows of \( C \) have the same sum). The information centrality scores are then equal to

\[
C_I = \frac{1}{T + \frac{S_T - 2S_R}{|V(G)|}}
\]

(recalling that the scores for any omitted vertices are 0).
In general, actors with higher information centrality are predicted to have greater control over the flow of information within a network; highly information-central individuals tend to have a large number of short paths to many others within the social structure. Because the raw centrality values can be difficult to interpret directly, rescaled values are sometimes preferred (see the rescale option). Though the use of path weights suggest information centrality as a possible replacement for closeness, the problem of inverting the $B$ matrix poses problems of its own; as with all such measures, caution is advised on disconnected or degenerate structures.

**Value**

A vector containing the centrality scores

**Note**

The theoretical maximum deviation used here is not obtained with the star network; rather, the maximum occurs for an empty graph with one complete dyad, which is the model used here.

**Author(s)**

David Barron (david.barron@jesus.ox.ac.uk)
Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

`evcent`, `bonpow`, `closeness`, `graphcent`, `centralization`

**Examples**

```r
#Generate some test data
dat<-rgraph(10,mode="graph")
#Compute information centrality scores
infocent(dat)
```

---

**interval.graph**

*Convert Spell Data to Interval Graphs*

**Description**

Constructs one or more interval graphs (and exchangeability vectors) from a set of spells.

**Usage**

```r
interval.graph(slist, type="simple", diag=FALSE)
```
**Arguments**

slist  
A spell list. This must consist of an nxmx3 array, with n being the number of actors, m being the maximum number of spells (one per row) and with the three columns of the last dimension containing a (categorical) spell type code, the time of spell onset (any units), and the time of spell termination (same units), respectively.

type  
One of “simple”, “overlap”, “fracxy”, “fracyx”, or “jntfrac”.

diag  
Include the dyadic entries?

**Details**

Given some ordering dimension T (usually time), a “spell” is defined as the interval between a specified onset and a specified termination (with onset preceding the termination). An interval graph, then, on spell set V, is $G = \{V, E\}$, where $\{i, j\} \in E$ iff there exists some point $t \in T$ such that $t \in i$ and $t \in j$. In more prosaic terms, an interval graph on a given spell set has each spell as a vertex, with vertices adjacent iff they overlap. Such structures are useful for quantifying life history data (where spells might represent marriages, periods of child custody/co-residence, periods of employment, etc.), organizational history data (where spells might reflect periods of strategic alliances, participation in a particular product market, etc.), task scheduling (with spells representing the dedication of a particular resource to a given task), etc. By giving complex historical data a graphic representation, it is possible to easily perform a range of analyses which would otherwise be difficult and/or impossible (see Butts and Pixley (2004) for examples).

In addition to the simple interval graph (described above), `interval.graph` can also generate valued interval graphs using a number of different edge definitions. This is controlled by the `type` argument, with edge values as follows:

1. simple: dichotomous coding based on simple overlap (i.e., $(x,y)=1$ iff x overlaps y)
2. overlap: edge value equals the total magnitude of the overlap between spells
3. fracxy: the $(x,y)$ edge value equals the fraction of the duration of y which is covered by x
4. fracyx: the $(x,y)$ edge value equals the fraction of the duration of x which is covered by y
5. jntfrac: edge value equals the total magnitude of the overlap between spells divided by the mean of the spells’ lengths

Note that “simple,” “overlap,” and “jntfrac” are symmetric relations, while “fracxy” and “fracyx” are directed. As always, the specific edge type used should reflect the application to which the interval graph is being put.

**Value**

A data frame containing:

- `graph`  
A graph stack containing the interval graphs
- `exchange.list`  
Matrix containing the vector of spell types associated with each interval graph

**Author(s)**

Carter T. Butts (buttsc@uci.edu)
References


Examples

```
is.connected      Is a Given Graph Connected?
```

Description

Returns TRUE iff the specified graph is connected.

Usage

```
is.connected(g, connected = "strong", comp.dist.precomp = NULL)
```

Arguments

- `g`: an adjacency matrix.
- `connected`: definition of connectedness to use; must be one of "strong", "weak", "unilateral", or "recursive".
- `comp.dist.precomp`: a `component.dist` object precomputed for the graph to be analyzed (optional).

Details

`is.connected` determines whether `g` is connected under the definition specified in `connected`.
(See `component.dist` for details.) Since `is.connected` is really just a wrapper for `component.dist`, an object created with the latter can be supplied (via `comp.dist.precomp`) to speed computation.

Value

TRUE iff `g` is connected, otherwise FALSE

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

`component.dist`, `components`
is.isolate

Examples

```r
#Generate two graphs:
g1<-rgraph(10,tp=0.1)
g2<-rgraph(10)

#Check for connectedness
is.connected(g1)  #Probably not
is.connected(g2)  #Probably so
```

### Description

Returns TRUE iff ego is an isolate in graph g of dat.

### Usage

```r
is.isolate(dat, ego, g=1, diag=FALSE)
```

### Arguments

- **dat**: A graph or graph stack
- **ego**: Index of the vertex (or a vector of vertices) to check
- **g**: Which graph should be examined?
- **diag**: Boolean indicating whether adjacency matrix diagonals (i.e., loops) contain meaningful data

### Details

In the valued case, any non-zero edge value is taken as sufficient to establish a tie.

### Value

A boolean value (or vector thereof) indicating isolate status

### Author(s)

Carter T. Butts  ⟨butts@uci.edu⟩

### References


### See Also

`isolates`, `add.isolates`
isolates

Examples

#Generate a test graph
g<-rgraph(20)
g[,4]<-0  #Create an isolate
g[4,]<-0

#Check for isolates
is.isolate(g,2)  #2 is almost surely not an isolate
is.isolate(g,4)  #4 is, by construction

isolates  List the Isolates in a Graph or Graph Stack

Description

Returns a list of the isolates in the graph or graph stack given by dat.

Usage

isolates(dat, diag=FALSE)

Arguments

dat  A graph or graph stack
diag  Boolean indicating whether adjacency matrix diagonals (i.e., loops) contain meaningful data

Value

A vector containing the isolates, or a list of vectors if more than one graph was specified

Author(s)

Carter T. Butts (butsc@uci.edu)

References


See Also

is.isolate, add.isolates

Examples

#Generate a test graph
g<-rgraph(20)
g[,4]<-0  #Create an isolate
g[4,]<-0

#List the isolates
isolates(g)
lab.optimize  Optimize a Bivariate Graph Statistic Across a Set of Accessible Permutations

Description

lab.optimize is the front-end to a series of heuristic optimization routines (see below), all of which seek to maximize/minimize some bivariate graph statistic (e.g., graph correlation) across a set of vertex relabelings.

Usage

lab.optimize(d1, d2, FUN, exchange.list=0, seek="min", opt.method=c("anneal", "exhaustive", "mc", "hillclimb", "gumbel"), ...)
lab.optimize.anneal(d1, d2, FUN, exchange.list=0, seek="min", prob.init=1, prob.decay=0.99, freeze.time=1000, full.neighborhood=TRUE, ...)
lab.optimize.exhaustive(d1, d2, FUN, exchange.list=0, seek="min", ...)
lab.optimize.gumbel(d1, d2, FUN, exchange.list=0, seek="min", draws=500, tol=1e-5, estimator="median", ...)
lab.optimize.hillclimb(d1, d2, FUN, exchange.list=0, seek="min", ...)
lab.optimize.mc(d1, d2, FUN, exchange.list=0, seek="min", draws=1000, ...)

Arguments

d1 A single adjacency matrix
d2 Another single adjacency matrix
FUN A function taking two graphs as its first two arguments, and returning a numeric value
exchange.list Information on which vertices are exchangeable (see below); this must be a single number, a vector of length n, or a nx2 matrix.
seek "min" if the optimizer should seek a minimum, or "max" if a maximum should be sought
opt.method The particular optimization method to use
prob.init Initial acceptance probability for a downhill move (lab.optimize.anneal only)
prob.decay The decay (cooling) multiplier for the probability of accepting a downhill move (lab.optimize.anneal only)
freeze.time Number of iterations at which the annealer should be frozen (lab.optimize.anneal only)
full.neighborhood Should all moves in the binary-exchange neighborhood be evaluated at each iteration? (lab.optimize.anneal only)
tol Tolerance for estimation of gumbel distribution parameters (lab.optimize.gumbel only)
lab.optimize

**estimator**  Gumbel distribution statistic to use as optimal value prediction; must be one of "mean", "median", or "mode" (lab.optimize.gumbel only)

**draws**  Number of draws to take for gumbel and mc methods

...  Additional arguments to FUN

**Details**

`lab.optimize` is the front-end to a family of routines for optimizing a bivariate graph statistic over a set of permissible relabelings (or equivalently, permutations). The accessible permutation set is determined by the `exchange.list` argument, which is dealt with in the following manner. First, `exchange.list` is expanded to fill an nx2 matrix. If `exchange.list` is a single number, this is trivially accomplished by replication; if `exchange.list` is a vector of length n, the matrix is formed by cbinding two copies together. If `exchange.list` is already an nx2 matrix, it is left as-is. Once the nx2 exchangeability matrix has been formed, it is interpreted as follows: columns refer to graphs 1 and 2, respectively; rows refer to their corresponding vertices in the original adjacency matrices; and vertices are taken to be theoretically exchangeable iff their corresponding exchangeability matrix values are identical. To obtain an unlabeled graph statistic (the default), then, one could simply let `exchange.list` equal any single number. To obtain the labeled statistic, one would use the vector 1:n.

Assuming a non-degenerate set of accessible permutations/relabelings, optimization proceeds via the algorithm specified in `opt.method`. The optimization routines which are currently implemented use a variety of different techniques, each with certain advantages and disadvantages. A brief summary of each is as follows:

1. **exhaustive search ("exhaustive"):** Under exhaustive search, the entire space of accessible permutations is combed for the global optimum. This guarantees a correct answer, but at a very high price: the set of all permutations grows with the factorial of the number of vertices, and even substantial exchangeability constraints are unlikely to keep the number of permutations from growing out of control. While exhaustive search is possible for small graphs, unlabeled structures of size approximately 10 or greater cannot be treated using this algorithm within a reasonable time frame.

   Approximate complexity: on the order of \( \prod_{i \in L} |V_i|! \), where \( L \) is the set of exchangeability classes.

2. **hill climbing ("hillclimb"):** The hill climbing algorithm employed here searches, at each iteration, the set of all permissible binary exchanges of vertices. If one or more exchanges are found which are superior to the current permutation, the best alternative is taken. If no superior alternative is found, then the algorithm terminates. As one would expect, this algorithm is guaranteed to terminate on a local optimum; unfortunately, however, it is quite prone to becoming "stuck" in suboptimal solutions. In general, hill climbing is not recommended for permutation search, but the method may prove useful in certain circumstances.

   Approximate complexity: on the order of \( |V(G)|^2 \) per iteration, total complexity dependent on the number of iterations.

3. **simulated annealing ("anneal"):** The (fairly simple) annealing procedure here employed proceeds as follows. At each iteration, the set of all permissible binary exchanges (if `full.neighborhood==TRUE`) or a random selection from this set is evaluated. If a superior option is identified, the best of these is chosen. If no superior options are found, then the algorithm chooses randomly from the set of alternatives with probability equal to the current temperature, otherwise retaining its prior solution. After each iteration, the current temperature is reduced by a factor equal to `prob.decay`; the initial temperature is set by `prob.init`. When a number of iterations equal to `freeze.time` have been completed, the algorithm “freezes.” Once “frozen,” the annealer hillclimbs from its present location until no improvement is found, and terminates.
At termination, the best permutation identified so far is utilized; this need not be the most recent position (though it sometimes is).

Simulated annealing is sometimes called “noisy hill climbing” because it uses the introduction of random variation to a hill climbing routine to avoid convergence to local optima; it works well on reasonably correlated search spaces with well-defined solution neighborhoods, and is far more robust than hill climbing algorithms. As a general rule, simulated annealing is recommended here for most graphs up to size approximately 50. At this point, computational complexity begins to become a serious barrier, and alternative methods may be more practical. Approximate complexity: on the order of \(|V(G)|^2 \cdot \text{freeze.time} | \text{if full.neighborhood}==\text{TRUE}, \text{otherwise complexity scales approximately linearly with freeze.time. This can be misleading, however, since failing to search the full neighborhood generally requires that freeze.time be greatly increased.}

4. blind monte carlo search (“mc”): Blind monte carlo search, as the name implies, consists of randomly drawing a sample of permutations from the accessible permutation set and selecting the best. Although this not such a bad option when A) a large fraction of points are optimal or nearly optimal and B) the search space is largely uncorrelated, these conditions do not seem to characterize most permutation search problems. Blind monte carlo search is not generally recommended, but it is provided as an option should it be desired (e.g., when it is absolutely necessary to control the number of permutations examined).

Approximate complexity: linear in \text{draws}.

5. extreme value estimation (“gumbel”): Extreme value estimation attempts to estimate a global optimum via stochastic modeling of the distribution of the graph statistic over the space of accessible permutations. The algorithm currently proceeds as follows. First, a random sample is taken from the accessible permutation set (as with monte carlo search, above). Next, this sample is used to fit an extreme value (gumbel) model; the gumbel distribution is the limiting distribution of the extreme values from samples under a continuous, unbounded distribution, and we use it here as an approximation. Having fit the model, an associated statistic (the mean, median, or mode as determined by \text{estimator}) is then used as an estimator of the global optimum.

Obviously, this approach has certain drawbacks. First of all, our use of the gumbel model in particular assumes an unbounded, continuous underlying distribution, which may or may not be approximately true for any given problem. Secondly, the inherent non-robustness of extremal problems makes the fact that our prediction rests on a string of approximations rather worrisome: our idea of the shape of the underlying distribution could be distorted by a bad sample, our parameter estimation could be somewhat off, etc., any of which could have serious consequences for our extremal prediction. Finally, the prediction which is made by the extreme value model is nonconstructive, in the sense that no permutation need have been found by the algorithm which induces the predicted value. On the bright side, this could allow one to estimate the optimum without having to find it directly; on the dark side, this means that the reported optimum could be a numerical chimera.

At this time, extreme value estimation should be considered experimental, and \text{is not recommended for use on substantive problems}. \text{lab.optimize.gumbel} is not guaranteed to work properly, or to produce intelligible results; this may eventually change in future revisions, or the routine may be scrapped altogether.

Approximate complexity: linear in \text{draws}.

This list of algorithms is itself somewhat unstable: some additional techniques (canonical labeling and genetic algorithms, for instance) may be added, and some existing methods (e.g., extreme value estimation) may be modified or removed. Every attempt will be made to keep the command format as stable as possible for other routines (e.g., \text{gscov, structdist}) which depend on \text{lab.optimize} to do their heavy-lifting. In general, it is not expected that the end-user will call...
lab.optimize directly; instead, most end-user interaction with these routines will be via the structural distance/covariance functions which used them.

**Value**

The estimated global optimum of FUN over the set of relabelings permitted by exchange.list

**Author(s)**

Carter T. Butts (butts@uci.edu)

**References**


**See Also**

gscov, gscor, structdist, sdmat

**Examples**

```r
#Generate a random graph and copy it
g<-rgraph(10)
g2<-rmperm(g)  #Permute the copy randomly

#Seek the maximum correlation
lab.optimize(g,g2,gcor,seek="max",opt.method="anneal",freeze.time=50,
prob.decay=0.9)

#These two don’t do so well...
lab.optimize(g,g2,gcor,seek="max",opt.method="hillclimb")
lab.optimize(g,g2,gcor,seek="max",opt.method="mc",draws=1000)
```

---

**lnam**  
*Fit a Linear Network Autocorrelation Model*

**Description**

*lnam* is used to fit linear network autocorrelation models. These include standard OLS as a special case, although *lm* is to be preferred for such analyses.

**Usage**

```r
lnam(y, x = NULL, W1 = NULL, W2 = NULL, theta.seed = NULL,
null.model = c("meanstd", "mean", "std", "none"), method = "BFGS",
control = list())
```
Arguments

y: a vector of responses.

x: a vector or matrix of covariates; if the latter, each column should contain a single covariate.

W1: a (possibly valued) adjacency matrix on the elements of y.

W2: another (possibly valued) adjacency matrix on the elements of y.

theta.seed: an optional seed value for optim.

null.model: the null model to be fit; must be one of "meanstd", "mean", "std", or "none".

method: method to be used with optim.

control: optional control parameters for optim.

Details

lnam fits the linear network autocorrelation model given by

\[ y = \rho_1 W_1 y + X \beta + e, \quad e = \rho_2 W_2 e + \nu \]

where y is a vector of responses, X is a covariate matrix, W1 and W2 are (possibly valued) adjacency matrices, and \( \nu \sim N(0, \sigma^2) \). Intuitively, \( \rho_1 \) is an “AR”-like parameter (parameterizing the autoregression of each y value on its neighbors in W1) while \( \rho_2 \) is a “MA”-like parameter (parameterizing the autocorrelation of each disturbance in y on its neighbors in W2). In general, the two models are distinct, and either or both effects may be selected by including the appropriate matrix arguments.

Model parameters are estimated by maximum likelihood, and asymptotic standard errors are provided as well; all of the above (and more) can be obtained by means of the appropriate print and summary methods. A plotting method is also provided, which supplies fit basic diagnostics for the estimated model. For purposes of comparison, fits may be evaluated against one of four null models:

1. meanstd: mean and standard deviation estimated (default).
2. mean: mean estimated; standard deviation assumed equal to 1.
3. std: standard deviation estimated; mean assumed equal to 0.
4. none: no parameters estimated; data assumed to be drawn from a standard normal density.

The default setting should be appropriate for the vast majority of cases, although the others may have use when fitting “pure” autoregressive models (e.g., without covariates). Although a major use of the lnam is in controlling for network autocorrelation within a regression context, the model is subtle and has a variety of uses. (See the references below for suggestions.)

Value

An object of class "lnam" containing the following elements:

y: the response vector used.

x: if supplied, the coefficient matrix.

W1: if supplied, the W1 matrix.

W2: if supplied, the W2 matrix.
model a code indicating the model terms fit.
infomat the estimated Fisher information matrix for the fitted model.
acvm the estimated asymptotic covariance matrix for the model parameters.
null.model a string indicating the null model fit.
lnlik.null the log-likelihood of $y$ under the null model.
df.null resid the residual degrees of freedom under the null model.
df.null the model degrees of freedom under the null model.
null.param parameter estimates for the null model.
lnlik.model the log-likelihood of $y$ under the fitted model.
df.model the model degrees of freedom.
df.residual the residual degrees of freedom.
df.total the total degrees of freedom.
rho1 if applicable, the MLE for $\rho_1$.
rho1.se if applicable, the asymptotic standard error for $\rho_1$.
rho2 if applicable, the MLE for $\rho_2$.
rho2.se if applicable, the asymptotic standard error for $\rho_2$.
sigma the MLE for $\sigma$.
sigma.se the standard error for $\sigma$.
beta if applicable, the MLE for $\beta$.
beta.se if applicable, the asymptotic standard errors for $\beta$.
fitted.values the fitted mean values.
residuals the residuals (response minus fitted); note that these correspond to $\hat{e}$ in the model equation, not $\hat{\nu}$.
disturbances the estimated disturbances, i.e., $\hat{\nu}$.
call the matched call.

**Note**

Actual optimization is performed by calls to `optim`. Information on algorithms and control parameters can be found via the appropriate man pages.

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

`lm`, `optim`
### lower.tri.remove

Remove the Lower Triangles of Adjacency Matrices in a Graph Stack

#### Description

Returns the input graph stack, with the lower triangle entries removed/replaced as indicated.

#### Usage

```r
lower.tri.remove(dat, remove.val=NA)
```

#### Arguments

- `dat`  
  A graph stack

- `remove.val`  
  The value with which to replace the existing lower triangles

#### Details

`lower.tri.remove` is simply a convenient way to apply `g[lower.tri(g)]<-remove.val` to an entire stack of adjacency matrices at once.

#### Value

The updated graph stack

#### Author(s)

Carter T. Butts (buttcsc@uci.edu)

#### References

Examples

```r
# Construct a simple, random example:
w1<-rgraph(100)  # Draw the AR matrix
w2<-rgraph(100)  # Draw the MA matrix
x<-matrix(rnorm(100*5),100,5)  # Draw some covariates
r1<-0.2          # Set the model parameters
r2<-0.1
sigma<-0.1
beta<-rnorm(5)
# Assemble y from its components:
u<-rnorm(100,0,sigma)  # Draw the disturbances
e<-qr.solve(diag(100)-r2*w2,nu)  # Draw the effective errors
y<-qr.solve(diag(100)-r1*w1,x%*%beta+e)  # Compute y

# Now, fit the autocorrelation model:
fit<-lnam(y,x,w1,w2)
summary(fit)
plot(fit)
```
lubness

Compute Graph LUBness Scores

Description

lubness takes a graph stack (dat) and returns the Krackhardt LUBness scores for the graphs selected by g.

Usage

lubness(dat, g=1:stackcount(dat))

Arguments

dat A graph or graph stack

g Index values for the graphs to be utilized; by default, all graphs are selected

Details

In the context of a directed graph $G$, two actors $i$ and $j$ may be said to have an upper bound iff there exists some actor $k$ such that directed $ki$ and $kj$ paths are belong to $G$. An upper bound $\ell$ is known as a least upper bound for $i$ and $j$ iff it belongs to at least one $ki$ and $kj$ path (respectively) for all $i, j$ upper bounds $k$; let $L(i, j)$ be an indicator which returns 1 iff such an $\ell$ exists, otherwise returning 0. Now, let $G_1, G_2, \ldots, G_n$ represent the weak components of $G$. For convenience, we denote the cardinalities of these graphs’ vertex sets by $|V(G)| = N$ and $|V(G_i)| = N_i$, $\forall i \in 1, \ldots, n$. Given this, the Krackhardt LUBness of $G$ is given by

$$1 - \frac{\sum_{i=1}^{n} \sum_{v_j, v_k \in V(G_i)} \left(1 - L(v_j, v_k)\right)}{\sum_{i=1}^{n} \frac{1}{2}(N_i - 1)(N_i - 2)}$$

Where all vertex pairs possess a least upper bound, Krackhardt’s LUBness is equal to 1; in general, it approaches 0 as this condition is broached. (This convergence is problematic in certain cases due to the requirement that we sum violations across components; where a graph contains no components of size three or greater, Krackhardt’s LUBness is not well-defined. lubness returns a NaN in these cases.)

LUBness is one of four measures (connectedness, efficiency, hierarchy, and lubness) suggested by Krackhardt for summarizing hierarchical structures. Each corresponds to one of four axioms which are necessary and sufficient for the structure in question to be an outtree; thus, the measures will be equal to 1 for a given graph iff that graph is an outtree. Deviations from unity can be interpreted in terms of failure to satisfy one or more of the outtree conditions, information which may be useful in classifying its structural properties.

See Also

lower.tri, upper.tri.remove, diag.remove

Examples

#Generate a random graph stack
g<-rgraph(3,5)
#Remove the lower triangles

g<-lower.tri.remove(g)
make.stochastic

Value
A vector of LUBness scores

Note
The four Krackhardt indices are, in general, nondegenerate for a relatively narrow band of size/density combinations (efficiency being the sole exception). This is primarily due to their dependence on the reachability graph, which tends to become complete rapidly as size/density increase. See Krackhardt (1994) for a useful simulation study.

Author(s)
Carter T. Butts (buttsc@uci.edu)

References

See Also
connectedness, efficiency, hierarchy, lubness, reachability

Examples
#Get LUBness scores for graphs of varying densities
lubness(rgraph(10,5,tprob=c(0.1,0.25,0.5,0.75,0.9)))

make.stochastic  

Make a Graph Stack Row, Column, or Row-column Stochastic

Description
Returns a graph stack in which each adjacency matrix in dat has been normalized to row stochastic, column stochastic, or row-column stochastic form, as specified by mode.

Usage
make.stochastic(dat, mode="rowcol", tol=0.005,
    maxiter=prod(dim(dat)) * 100, anneal.decay=0.01, errpow=1)

Arguments
dat A graph or graph stack
mode One of “row,” “col,” or “rowcol”
tol Tolerance parameter for the row-column normalization algorithm
maxiter Maximum iterations for the row-column normalization algorithm
anneal.decay Probability decay factor for the row-column annealer
errpow Power to which absolute row-column normalization errors should be raised for the annealer (i.e., the penalty function)
Row and column stochastic matrices are those whose rows and columns sum to 1 (respectively). These are quite straightforwardly produced here by dividing each row (or column) by its sum. Row-column stochastic matrices, by contrast, are those in which each row and each column sums to 1. Here, we try to produce row-column stochastic matrices whose values are as close in proportion to the original data as possible by means of an annealing algorithm. This is probably not optimal in the long term, but the results seem to be consistent where row-column stochasticization of the original data is possible (which it is not in all cases).

The stochasticized adjacency matrices

Rows or columns which sum to 0 in the original data will generate undefined results. This can happen if, for instance, your input graphs contain in- or out-isolates.

Carter T. Butts (buttsc@uci.edu)

Examples

# Generate a test matrix
g<-rgraph(15)

# Make it row stochastic
make.stochastic(g,mode="row")

# Make it column stochastic
make.stochastic(g,mode="col")

# (Try to) make it row-column stochastic
make.stochastic(g,mode="rowcol")
Arguments

- **dat**: A graph or graph stack
- **g**: A vector indicating which elements of **dat** should be analyzed; by default, all graphs are included

Details

The mutuality of a digraph G is defined as the number of complete dyads (i.e., i<->j) within G. (Compare this to dyadic reciprocity, the fraction of dyads within G which are symmetric.) Mutuality is commonly employed as a measure of reciprocal tendency within the p* literature; although mutuality can be very hard to interpret in practice, it is much better behaved than many alternative measures.

Value

One or more mutuality scores

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

grecip

Examples

```r
# Create some random graphs
g <- rgraph(15, 3)

# Get mutuality and reciprocity scores
mutuality(g)
grecip(g)  # Compare with mutuality
```

Description

*netcancor* finds the canonical correlation(s) between the graph sets x and y, testing the result using either conditional uniform graph (CUG) or quadratic assignment procedure (QAP) null hypotheses.

Usage

```r
netcancor(y, x, mode="digraph", diag=FALSE, nullhyp="cugtie", reps=1000)
```
Arguments

**y**  
First data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Missing values are not allowed.

**x**  
Second data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Missing values are not allowed.

**mode**  
String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.

**diag**  
Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

**nullhyp**  
String indicating the particular null hypothesis against which to test the observed estimands. A value of "cug" implies a conditional uniform graph test (see cugtest) controlling for order only; "cugden" controls for both order and tie probability; "cugtie" controls for order and tie distribution (via bootstrap); and "qap" implies that the QAP null hypothesis (see qaptest) should be used.

**reps**  
Integer indicating the number of draws to use for quantile estimation. (Relevant to the null hypothesis test only - the analysis itself is unaffected by this parameter.) Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles.

Details

The netcancor routine is actually a front-end to the cancor routine for computing canonical correlations between sets of vectors. netcancor itself vectorizes the network variables (as per its graph type) and manages the appropriate null hypothesis tests; the actual canonical correlation is handled by cancor. Canonical correlation itself is a multivariate generalization of the product-moment correlation. Specifically, the analysis seeks linear combinations of the variables in y which are well-explained by linear combinations of the variables in x. The network version of this technique is performed elementwise on the adjacency matrices of the graphs in question; as usual, the result should be interpreted with an eye to the relationship between the type of data used and the assumptions of the underlying model.

Intelligent printing and summarizing of netcancor objects is provided by print.netcancor and summary.netcancor.

Value

An object of class netcancor with the following properties:

**xdist**  
Array containing the distribution of the X coefficients under the null hypothesis test.

**ydist**  
Array containing the distribution of the Y coefficients under the null hypothesis test.

**cdist**  
Array containing the distribution of the canonical correlation coefficients under the null hypothesis test.

**cor**  
Vector containing the observed canonical correlation coefficients.

**xcoef**  
Vector containing the observed X coefficients.
ycoef        Vector containing the observed Y coefficients.
cpgreq      Vector containing the estimated upper tail quantiles (p>=obs) for the observed canonical correlation coefficients under the null hypothesis.
cpleeq      Vector containing the estimated lower tail quantiles (p<=obs) for the observed canonical correlation coefficients under the null hypothesis.
xpgreq      Matrix containing the estimated upper tail quantiles (p>=obs) for the observed X coefficients under the null hypothesis.
xpleeq      Matrix containing the estimated lower tail quantiles (p<=obs) for the observed X coefficients under the null hypothesis.
ypgreq      Matrix containing the estimated upper tail quantiles (p>=obs) for the observed Y coefficients under the null hypothesis.
ypleeq      Matrix containing the estimated lower tail quantiles (p<=obs) for the observed Y coefficients under the null hypothesis.
cnames      Vector containing names for the canonical correlation coefficients.
xnames      Vector containing names for the X vars.
ynames      Vector containing names for the Y vars.
xcenter      Values used to adjust the X variables.
xcenter      Values used to adjust the Y variables.
nullhyp      String indicating the null hypothesis employed.

Note

This will eventually be replaced with a superior cancort procedure with more interpretable output; the new version will handle arbitrary labeling as well.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

gcor, cugtest, qaptest, cancort

Examples

#Generate a valued seed structure
cv<-matrix(rnorm(100),nrow=10,ncol=10)
#Produce two sets of valued graphs
x<-array(dim=c(3,10,10))
x[1,,]<-3*cv+matrix(rnorm(100,0,0.1),nrow=10,ncol=10)
x[2,,]<<-1*cv+matrix(rnorm(100,0,0.1),nrow=10,ncol=10)
x[3,,]<-x[1,,]+2*x[2,,]+5*cv+matrix(rnorm(100,0,0.1),nrow=10,ncol=10)
y<-array(dim=c(2,10,10))
y[1,,]<<-5*cv+matrix(rnorm(100,0,0.1),nrow=10,ncol=10)
y[2,,]<<-2*cv+matrix(rnorm(100,0,0.1),nrow=10,ncol=10)
#Perform a canonical correlation analysis
netlm

107

nc<-netcancor(y,x,reps=100)
summary(nc)

netlm

Linear Regression for Network Data

Description

netlm regresses the network variable in y on the network variables in stack x using ordinary least squares. The resulting fits (and coefficients) are then tested against the indicated null hypothesis.

Usage

netlm(y, x, mode="digraph", diag=FALSE, nullhyp="cugtie", reps=1000)

Arguments

y Dependent network variable. This should be a matrix, for obvious reasons; NAs are allowed, but dichotomous data is strongly discouraged due to the assumptions of the analysis.

x Data array containing the stack of independent network variables. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Note that NAs are permitted, as is dichotomous data.

mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.

diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

nullhyp String indicating the particular null hypothesis against which to test the observed estimands. A value of "cug" implies a conditional uniform graph test (see cugtest) controlling for order only; "cugden" controls for both order and tie probability; "cugtie" controls for order and tie distribution (via bootstrap); and "qap" implies that the QAP null hypothesis (see qaptest) should be used.

reps Integer indicating the number of draws to use for quantile estimation. ( Relevant to the null hypothesis test only - the analysis itself is unaffected by this parameter.) Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, reps=1000.

Details

Unsurprisingly, netlm is really a front-end to the built-in lm routine. netlm handles vectorization and null hypothesis testing; the actual fitting is taken care of by lm.

Network regression using OLS is directly analogous to standard OLS regression elementwise on the appropriately vectorized adjacency matrices of the networks involved. In particular, the network regression attempts to fit the model:

\[ A_y = b_0 A_1 + b_1 A_{x_1} + b_2 A_{x_2} + \ldots + Z \]

where \( A_y \) is the dependent adjacency matrix, \( A_{x_i} \) is the ith independent adjacency matrix, \( A_1 \) is an n x n matrix of 1’s, and \( Z \) is an n x n matrix of independent normal random variables with mean zero and unit variance.
0 and variance $\sigma^2$. Clearly, this model is nonoptimal when $A_y$ is dichotomous (or, for that matter, categorical in general); an alternative such as netlogit should be employed in such cases. (Note that netlm will still attempt to fit such data...the user should consider him or herself to have been warned.)

Null hypothesis tests for the network regression are handled using either the conditional uniform graph hypothesis (the default) or QAP. See the help pages for these tests for a fuller description of each. Reasonable printing and summarizing of netlm objects is provided by print.netlm and summary.netlm, respectively. No plot methods exist at this time, alas.

Value

An object of class netlm

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

lm, netlogit

Examples

#Create some input graphs
x<-rgraph(20,4)

#Create a response structure
y<-x[1,,]+4*x[2,,]+2*x[3,,] #Note that the fourth graph is unrelated

#Fit a netlm model
nl<-netlm(y,x,reps=100)

#Examine the results
summary(nl)

---

**netlogit**  
*Logistic Regression for Network Data*

Description

netlogit performs a logistic regression of the network variable in $y$ on the network variables in stack $x$. The resulting fits (and coefficients) are then tested against the indicated null hypothesis.
**Usage**

```r
netlogit(y, x, mode="digraph", diag=FALSE, nullhyp="cugtie", 
reps=1000)
```

**Arguments**

- `y`: Dependent network variable. This should be a matrix, for obvious reasons; NAs are allowed, and the data should be dichotomous.
- `x`: Data array containing the stack of independent network variables. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Note that NAs are permitted, as is dichotomous data.
- `mode`: String indicating the type of graph being evaluated. "Digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. `mode` is set to "digraph" by default.
- `diag`: Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. `diag` is FALSE by default.
- `nullhyp`: String indicating the particular null hypothesis against which to test the observed estimands. A value of "cug" implies a conditional uniform graph test (see `cugtest`) controlling for order only; "cugden" controls for both order and tie probability; "cugtie" controls for order and tie distribution (via bootstrap); and "qap" implies that the QAP null hypothesis (see `qaptest`) should be used.
- `reps`: Integer indicating the number of draws to use for quantile estimation. (Relevant to the null hypothesis test only – the analysis itself is unaffected by this parameter.) Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, `reps`=1000.

**Details**

`netlogit` is primarily a front-end to the built-in `glm` routine. `netlogit` handles vectorization, sets up `glm` options, and deals with null hypothesis testing; the actual fitting is taken care of by `glm`.

Logistic network regression using is directly analogous to standard logistic regression elementwise on the appropriately vectorized adjacency matrices of the networks involved. As such, it is often a more appropriate model for fitting dichotomous response networks than is linear network regression.

Null hypothesis tests for logistic network regression are handled using either the conditional uniform graph hypothesis (the default) or QAP. See the help pages for these tests for a fuller description of each. Reasonable printing and summarizing of `netlogit` objects is provided by `print.netlogit` and `summary.netlogit`, respectively. No plot methods exist at this time.

**Value**

An object of class `netlogit`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**

See Also

`glm`, `netlm`

Examples

```r
# Create some input graphs
x <- rgraph(20, 4)

# Create a response structure
y.l <- x[,1,] + 4*x[2,] + 2*x[3,]  # Note that the fourth graph is unrelated
y.p <- apply(y.l, c(1, 2), function(a) {1/(1 + exp(-a))})
y <- rgraph(20, tprob = y.p)

# Fit a netlogit model
nl <- netlogit(y, x, reps = 100)

# Examine the results
summary(nl)
```

npostpred

Take Posterior Predictive Draws for Functions of Networks

Description

`npostpred` takes a list or data frame, `b`, and applies the function `FUN` to each element of `b`'s `net` member.

Usage

```r
npostpred(b, FUN, ...)
```

Arguments

- `b` A list or data frame containing posterior network draws; these draws must take the form of a graph stack, and must be the member of `b` referenced by "net"
- `FUN` Function for which posterior predictive is to be estimated
- `...` Additional arguments to `FUN`

Details

Although created to work with `bbnam`, `npostpred` is quite generic. The form of the posterior draws will vary with the output of `FUN`; since invocation is handled by `apply`, check there if unsure.

Value

A series of posterior predictive draws

Author(s)

Carter T. Butts (buttsc@uci.edu)
References


See Also

bbnam

Examples

```r
#Create some random data
g<-rgraph(5)
g.p<-0.8*g+0.2*(1-g)
dat<-rgraph(5,5,tprob=g.p)

#Define a network prior
pnet<-matrix(ncol=5,nrow=5)
pnet[,]<-0.5
#Define em and ep priors
pem<-matrix(nrow=5,ncol=2)
pem[,1]<-3
pem[,2]<-5
pep<-matrix(nrow=5,ncol=2)
pep[,1]<-3
pep[,2]<-5

#Draw from the posterior
b<-bbnam(dat,model="actor",nprior=pnet,emprior=pem,epprior=pep,
burntime=100,draws=100)
#Plot a summary of the posterior predictive of reciprocity
hist(npostpred(b,grecip))
```

nties

*nnties* Find the Number of Possible Ties in a Given Graph or Stack

Description

*nties* returns the number of possible edges in each element of *dat*, given *mode* and *diag*.

Usage

*nties(dat, mode="digraph", diag=FALSE)*

Arguments

- **dat**: A graph or stack thereof
- **mode**: One of “digraph”, “graph”, and “hgraph”
- **diag**: A boolean indicating whether or not diagonal entries (loops) should be treated as valid data; ignored for hypergraphic (“hgraph”) data

Details

*nties* is used primarily to automate maximum edge counts for use with normalization routines.
Value
The number of possible edges, or a vector of the same

Note
For two-mode (hypergraphic) data, the value returned isn’t technically the number of edges per se, but rather the number of edge memberships.

Author(s)
Carter T. Butts ⟨buttsc@uci.edu⟩

Examples
# How many possible edges in a loopless digraph of order 15?
nties(rgraph(15), diag=FALSE)

---

numperm **Get the nth Permutation Vector by Periodic Placement**

Description
numperm implicitly numbers all permutations of length `olength`, returning the `permnum`th of these.

Usage
numperm(`olength`, `permnum`)

Arguments
- `olength`: The number of items to permute
- `permnum`: The number of the permutation to use (in `1:olength!`)

Details
The `n!` permutations on `n` items can be deterministically ordered via a factorization process in which there are `n` slots for the first element, `n-1` for the second, and `n-i` for the `i`th. This fact is quite handy if you want to visit each permutation in turn, or if you wish to sample without replacement from the set of permutations on some number of elements: one just enumerates or samples from the integers on `[1,n!]`, and then find the associated permutation. `numperm` performs exactly this last operation, returning the `permnum`th permutation on `olength` items.

Value
A permutation vector

Note
Permutation search is central to the estimation of structural distances, correlations, and covariances on partially labeled graphs. `numperm` is hence used by `structdist`, `gscor`, `gscov`, etc.
**plot.bbnam**

**Author(s)**

Carter T. Butts (buttc@uci.edu)

**See Also**

rperm, rmperm

**Examples**

```r
# Draw a graph
g <- rgraph(5)

# Permute the rows and columns
p.1 <- numperm(5, 1)
p.2 <- numperm(5, 2)
p.3 <- numperm(5, 3)
g[p.1, p.1]
g[p.2, p.2]
g[p.3, p.3]
```

**Description**

Generates various plots of posterior draws from the bbnam model.

**Usage**

```r
plot.bbnam(x, mode="density", intlines=TRUE, ...)
```

**Arguments**

- `x` : A bbnam object
- `mode` : “density” for kernel density estimators of posterior marginals; otherwise, histograms are used
- `intlines` : Plot lines for the 0.9 central posterior probability intervals?
- `...` : Additional arguments to `plot`

**Details**

`plot.bbnam` provides plots of the estimated posterior marginals for the criterion graph and error parameters (as appropriate). Plotting may run into difficulties when dealing with large graphs, due to the problem of getting all of the various plots on the page; the routine handles these issues reasonably intelligently, but there is doubtless room for improvement.

**Value**

None
plot.blockmodel

Description

Displays a plot of the blocked data matrix, given a blockmodel object.

Usage

plot.blockmodel(x, ...)

Arguments

x An object of class blockmodel

... Further arguments passed to or from other methods
**Details**

Plots of the blocked data matrix (i.e., the data matrix with rows and columns permuted to match block membership) can be useful in assessing the strength of the block solution (particularly for clique detection and/or regular equivalence).

**Value**

None

**Author(s)**

Carter T. Butts (butsc@uci.edu)

**References**


**See Also**

`blockmodel`, `plot.sociomatrix`

**Examples**

```r
#Create a random graph with _some_ edge structure
g.p<-sapply(runif(20,0,1),rep,20) #Create a matrix of edge probabilities
g<-rgraph(20,tprob=g.p) #Draw from a Bernoulli graph distribution

#Cluster based on structural equivalence
eq<-equiv.clust(g)

#Form a blockmodel with distance relaxation of 10
b<-blockmodel(g,eq,h=10)
plot(b) #Plot it
```

---

**plot.cugtest**  
Plotting for cugtest Objects

**Description**

Plots the distribution of a CUG test statistic.

**Usage**

```r
plot.cugtest(x, mode="density", ...)
```
plot.equiv.clust

Arguments

x
A cugtest object

mode
"density" for kernel density estimation, "hist" for histogram

... Additional arguments to plot

Details

In addition to the quantiles associated with a CUG test, it is often useful to examine the form of the
distribution of the test statistic. plot.cugtest facilitates this.

Value

None

Author(s)

Carter T. Butts (butsc@uci.edu)

References

Graph-Level Indices.” Social Networks, 21(3), 239-267.

See Also

cugtest

Examples

#Draw two random graphs, with different tie probabilities
dat<-rgraph(20,2,tprob=c(0.2,0.8))

#Is their correlation higher than would be expected, conditioning
#only on size?
cug<-cugtest(dat,gcor,cmode="order")
summary(cug)
plot(cug)

#Now, let's try conditioning on density as well.
cug<-cugtest(dat,gcor)
plot(cug)

plot.equiv.clust

Plot an equiv.clust Object

Description

Plots a hierarchical clustering of node positions as generated by equiv.clust.

Usage

plot.equiv.clust(x, labels=x$plabels, ...)
**plot.equiv.clust**

**Arguments**

- **x**: An `equiv.clust` object
- **labels**: A vector of vertex labels
- **...**: Additional arguments to `plot.hclust`

**Details**

`plot.equiv.clust` is actually a front-end to `plot.hclust`; see the latter for more additional documentation.

**Value**

None.

**Author(s)**

Carter T. Butts (butsc@uci.edu)

**References**


**See Also**

`equiv.clust`, `plot.hclust`

**Examples**

```r
# Create a random graph with some edge structure
r <- rgraph(20, tprob = g.p)  # Draw from a Bernoulli graph
# Create a matrix of edge probabilities

eq <- equiv.clust(g)  # Cluster based on structural equivalence
plot(eq)
```

```r
# Create a random graph with _some_ edge structure
g.p <- sapply(runif(20, 0, 1), rep, 20)  # Create a matrix of edge
# probabilities

g <- rgraph(20, tprob = g.p)  # Draw from a Bernoulli graph
# distribution
```
**plot.lnam**  
*Plotting for lnam Objects*

Description
Generates various diagnostic plots for `lnam` objects.

Usage
```
plot.lnam(x, ...)  
```

Arguments
- `x` an object of class `lnam`.
- `...` additional arguments to `plot`.

Value
None

Author(s)
Carter T. Butts (buttsc@uci.edu)

See Also
- `lnam`

---

**plot.qaptest**  
*Plotting for qaptest Objects*

Description
Plots the Distribution of a QAP Test Statistic.

Usage
```
plot.qaptest(x, mode="density", ...)  
```

Arguments
- `x` A `qaptest` object
- `mode` “density” for kernel density estimation, “hist” for histogram
- `...` Additional arguments to `plot`.

Details
In addition to the quantiles associated with a QAP test, it is often useful to examine the form of the distribution of the test statistic. `plot.qaptest` facilitates this.
**plot.sociomatrix**

**Value**

None

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

qaptest

**Examples**

```
#Generate three graphs
g <- array(dim=c(3,10,10))
g[1,,]<-rgraph(10)
g[2,,]<-rgraph(10,tprob=g[1,,]*0.8)
g[3,,]<-1; g[3,1,2]<-0 #This is nearly a clique

#Perform qap tests of graph correlation
q.12<-qaptest(g,gcor,g1=1,g2=2)
q.13<-qaptest(g,gcor,g1=1,g2=3)

#Examine the results
summary(q.12)
plot(q.12)
summary(q.13)
plot(q.13)
```

---

**plot.sociomatrix**  *Plot Matrices Using a Color/Intensity Grid*

**Description**

Plots a matrix, `m`, associating the magnitude of the i,jth cell of `m` with the color of the i,jth cell of an `nrow(m)` by `ncol(m)` grid.

**Usage**

```
plot.sociomatrix(x, labels=list(seq(1:dim(x)[1]), seq(1:dim(x)[2])), drawlab=TRUE, diaglab=TRUE, ...)
```
potscalered.mcmc

Compute Gelman and Rubin’s Potential Scale Reduction Measure for a Markov Chain Monte Carlo Simulation

Description

Computes Gelman and Rubin’s (simplified) measure of scale reduction for draws of a single scalar estimand from parallel MCMC chains.

Usage

potscalered.mcmc(psi)

Arguments

psi An nxm matrix, with columns corresponding to chains and rows corresponding to iterations.
Details

The Gelman and Rubin potential scale reduction ($\hat{R}$) provides an ANOVA-like comparison of the between-chain to within-chain variance on a given scalar estimand; the disparity between these gives an indication of the extent to which the scale of the simulated distribution can be reduced via further sampling. As the parallel chains converge $\hat{R}$ approaches 1 (from above), and it is generally recommended that values of 1.2 or less be obtained before a series of draws can be considered well-mixed. (Even so, one should ideally examine other indicators of chain mixing, and verify that the properties of the draws are as they should be. There is currently no fool-proof way to verify burn-in of an MCMC, but using multiple indicators should help one avoid falling prey to the idiosyncrasies of any one index.)

Note that the particular estimators used in the $\sqrt{\hat{R}}$ formulation are based on normal-theory results, and as such have been criticized vis a vis their behavior on other distributions. Where simulating distributions whose properties differ greatly from the normal, an alternative form of the measure using robust measures of scale (e.g., the IQR) may be preferable.

Value

The potential scale reduction measure

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

bbnam

Examples

prestige

Calculate the Vertex Prestige Scores

Description

prestige takes a graph stack (dat) and returns the prestige scores of positions within one graph (indicated by nodes and g, respectively). Depending on the specified mode, prestige based on any one of a number of different definitions will be returned. This function is compatible with centralization, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by centralization to normalize the observed centralization score).
prestige

Usage

prestige(dat, g=1, nodes=c(1:dim(dat)[2]), gmode="digraph", diag=FALSE, cmode="indegree", tmaxdev=FALSE, rescale=FALSE, tol=1e-07)

Arguments

dat  Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).
g  Integer indicating the index of the graph for which centralities are to be calculated. By default, g==1.

nodes  List indicating which nodes are to be included in the calculation. By default, all nodes are included.

gmode  String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. gmode is set to "digraph" by default.

diag  Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

cmode  One of "indegree", "indegree.rownorm", "indegree.rowcolnorm", "eigenvector", "eigenvector.rownorm", "eigenvector.colnorm", "eigenvector.rowcolnorm", "domain", or "domain.proximity"

tmaxdev  Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.

rescale  If true, centrality scores are rescaled such that they sum to 1.

tol  Currently ignored

Details

"Prestige" is the name collectively given to a range of centrality scores which focus on the extent to which one is nominated by others. The definitions supported here are as follows:

1. indegree: indegree centrality
2. indegree.rownorm: indegree within the row-normalized graph
3. indegree.rowcolnorm: indegree within the row-column normalized graph
4. eigenvector: eigenvector centrality within the transposed graph (i.e., incoming ties recursively determine prestige)
5. eigenvector.rownorm: eigenvector centrality within the transposed row-normalized graph
6. eigenvector.colnorm: eigenvector centrality within the transposed column-normalized graph
7. eigenvector.rowcolnorm: eigenvector centrality within the transposed row/column-normalized graph
8. domain: indegree within the reachability graph (Lin’s unweighted measure)
9. domain.proximity: Lin’s proximity-weighted domain prestige

Note that the centralization of prestige is simply the extent to which one actor has substantially greater prestige than others; the underlying definition is the same.
Value

A vector of prestige scores

Warning

Making adjacency matrices doubly stochastic (row-column normalization) is not guaranteed to work. In general, be wary of attempting to try normalizations on graphs with degenerate rows and columns.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

centralization

Examples

g<-rgraph(10) #Draw a random graph with 10 members
prestige(g,cmode="domain") #Compute domain prestige scores

---

**print.bayes.factor**  
*Printing for Bayes Factor Objects*

Description

Prints a quick summary of a Bayes Factor object.

Usage

`print.bayes.factor(x, ...)`

Arguments

- `x` An object of class `bayes.factor`
- `...` Further arguments passed to or from other methods

Value

None

Author(s)

Carter T. Butts (butts@uci.edu)
See Also

bbnam bf

Examples

print.bbnam(x, ...)

Arguments

x A bbnam object
...
Further arguments passed to or from other methods

Value

None

Author(s)

Carter T. Butts (butsc@uci.edu)

See Also

bbnam

Examples
print.blockmodel  Printing for blockmodel Objects

Description
Prints a quick summary of a blockmodel object.

Usage
print.blockmodel(x, ...)

Arguments
x  An object of class blockmodel
... Further arguments passed to or from other methods

Value
None

Author(s)
Carter T. Butts (butsc@uci.edu)

See Also
blockmodel

Examples

print.cugtest  Printing for cugtest Objects

Description
Prints a quick summary of objects produced by cugtest.

Usage
print.cugtest(x, ...)

Arguments
x  An object of class cugtest
... Further arguments passed to or from other methods

Value
None.
Author(s)

Carter T. Butts ⟨butts@uci.edu⟩

See Also
cugtest

Examples

print.lnam(x, digits = max(3, getOption("digits") - 3), ...)

Description

Prints an object of class lnam

Usage

print.lnam(x, digits = max(3, getOption("digits") - 3), ...)

Arguments

x an object of class lnam.
digits number of digits to display.
... additional arguments.

Value

None.

Author(s)

Carter T. Butts ⟨butts@uci.edu⟩

See Also

lnam
print.netcancor  

Prints a quick summary of objects produced by netcancor.

Usage

print.netcancor(x, ...)

Arguments

x  An object of class netcancor
...

Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts  (buttsc@uci.edu)

See Also

netcancor

Examples

print.netlm  

Printing for netlm Objects

Description

Prints a quick summary of objects produced by netlm

Usage

print.netlm(x, ...)

Arguments

x  An object of class netlm
...

Further arguments passed to or from other methods

Value
Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

See Also

netlm

Examples

```r
print.netlogit
```

Description

Prints a quick summary of objects produced by `netlogit`.

Usage

```r
print.netlogit(x, ...)
```

Arguments

- `x` An object of class `netlogit`
- `...` Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

See Also

- `netlogit`

Examples
print.qaptest  Printing for qaptest Objects

Description

Prints a quick summary of objects produced by qaptest.

Usage

print.qaptest(x, ...)

Arguments

x  An object of class qaptest
...

Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts  ⟨buttsc@uci.edu⟩

See Also

qaptest

Examples

print.summary.bayes.factor  Printing for summary.bayes.factor Objects

Description

Prints an object of class summary.bayes.factor.

Usage

print.summary.bayes.factor(x, ...)

Arguments

x  An object of class summary.bayes.factor
...

Further arguments passed to or from other methods
print.summary.bbnam

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

summary.bayes.factor

Examples

print.summary.bbnam

Printing for summary.bbnam Objects

Description

Prints an object of class summary.bbnam.

Usage

print.summary.bbnam(x, ...)

Arguments

x            An object of class summary.bbnam
          ...
          Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

bbnam

Examples
print.summary.blockmodel

Printing for summary.blockmodel Objects

Description
Prints an object of class summary.blockmodel.

Usage
print.summary.blockmodel(x, ...)

Arguments
x  An object of class summary.blockmodel
...  Further arguments passed to or from other methods

Value

Author(s)
Carter T. Butts (butts@uci.edu)

See Also
summary.blockmodel

Examples

print.summary.cugtest

Printing for summary.cugtest Objects

Description
Prints an object of class summary.cugtest.

Usage
print.summary.cugtest(x, ...)

Arguments
x  An object of class summary.cugtest
...  Further arguments passed to or from other methods
Value

Author(s)
Carter T. Butts ⟨buttsc@uci.edu⟩

See Also
summary.cugtest

Examples

print.summary.lnam  Printing for summary.lnam Objects

Description
Prints an object of class summary.lnam.

Usage

print.summary.lnam(x, digits = max(3, getOption("digits") - 3),
     signif.stars = getOption("show.signif.stars"), ...)

Arguments

  x  an object of class summary.lnam.
  digits  number of digits to display.
  signif.stars  show significance stars?
  ...  additional arguments.

Value
None

Author(s)
Carter T. Butts ⟨buttsc@uci.edu⟩

See Also
summary.lnam, lnam
print.summary.netcancor

Printing for summary.netcancor Objects

Description

Prints an object of class summary.netcancor.

Usage

print.summary.netcancor(x, ...)

Arguments

x An object of class summary.netcancor
...

Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

summary.netcancor

Examples

print.summary.netlm

Printing for summary.netlm Objects

Description

Prints an object of class summary.netlm.

Usage

print.summary.netlm(x, ...)

Arguments

x An object of class summary.netlm
...

Further arguments passed to or from other methods
print.summary.netlogit

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

summary.netlm

Examples

print.summary.netlogit

Printing for summary.netlogit Objects

Description

Prints an object of class summary.netlogit.

Usage

print.summary.netlogit(x, ...)

Arguments

x An object of class summary.netlogit
... Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

summary.netlogit

Examples
Description

Prints an object of class summary.qaptest.

Usage

\texttt{print.summary.qaptest(x, ...)}

Arguments

\texttt{x} \hspace{1cm} An object of class summary.qaptest
\texttt{...} \hspace{1cm} Further arguments passed to or from other methods

Value

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

\texttt{summary.qaptest}

Examples

\texttt{pstar}

\hspace{1cm} \textit{Fit a p*/ERG Model Using a Logistic Approximation}

Description

Fits a p*/ERG model to the adjacency matrix in \texttt{dat} containing the effects listed in \texttt{effects}. The result is returned as a \texttt{glm} object.

Usage

Arguments

- **dat**: A single adjacency matrix
- **effects**: A vector of strings indicating which effects should be fit
- **attr**: A matrix whose columns contain individual attributes (one row per vertex) whose differences should be used as supplemental predictors
- **memb**: A matrix whose columns contain group memberships whose categorical similarities (same group/not same group) should be used as supplemental predictors
- **diag**: A boolean indicating whether or not diagonal entries (loops) should be counted as meaningful data
- **mode**: "digraph" if `dat` is directed, else "graph"

Details

$p^*$ (also called the Exponential Random Graph (ERG) family) is an exponential family specification for network data. Under $p^*$, it is assumed that

$$p(G = g) \propto \exp(\beta_0 \gamma_0(g) + \beta_1 \gamma_1(g) + \ldots)$$

for all $g$, where the betas represent real coefficients and the gammas represent functions of $g$. Unfortunately, the unknown normalizing factor in the above expression makes evaluation difficult in the general case. One solution to this problem is to operate instead on the edgewise log odds; in this case, the $p^*$ can be approximated by a logistic regression of each edge on the differences in the gamma scores induced by the presence and absence of said edge in the graph (conditional on all other edges). It is this approximation (known as autologistic regression, or maximum pseudo-likelihood estimation) which is employed here.

Using the `effects` argument, a range of different potential parameters can be estimated. The network measure associated with each is, in turn, the edge-perturbed difference in:

1. **choice**: the number of edges in the graph (acts as a constant)
2. **mutuality**: the number of reciprocated dyads in the graph
3. **density**: the density of the graph
4. **reciprocity**: the edgewise reciprocity of the graph
5. **stransitivity**: the strong transitivity of the graph
6. **wtransitivity**: the weak transitivity of the graph
7. **stranstri**: the number of strongly transitive triads in the graph
8. **wtranstri**: the number of weakly transitive triads in the graph
9. **outdegree**: the outdegree of each actor (|V| parameters)
10. **indegree**: the indegree of each actor (|V| parameters)
11. **betweenness**: the betweenness of each actor (|V| parameters)
12. **closeness**: the closeness of each actor (|V| parameters)
13. **degcentralization**: the Freeman degree centralization of the graph
14. **betcentralization**: the betweenness centralization of the graph
15. **clocentralization**: the closeness centralization of the graph
16. **connectedness**: the Krackhardt connectedness of the graph
17. **hierarchy**: the Krackhardt hierarchy of the graph
18. **efficiency**: the Krackhardt efficiency of the graph
19. lubness: the Krackhardt LUBness of the graph

(Note that some of these do differ somewhat from the common p* parameter formulation, e.g. quantities such as density and reciprocity are computed as per the gden and grecip functions rather than via the unnormalized "choice" and "mutual" quantities one often finds in the p* literature.) Please do not attempt to use all effects simultaneously!!! In addition to the above, the user may specify a matrix of individual attributes whose absolute dyadic differences are to be used as predictors, as well as a matrix of individual memberships whose dyadic categorical similarities (same/different) are used in the same manner.

Although the p* framework is quite versatile in its ability to accommodate a range of structural predictors, it should be noted that the substantial collinearity of many of the standard p* predictors can lead to very unstable model fits. Measurement and specification errors compound this problem; thus, it is somewhat risky to use p* in an exploratory capacity (i.e., when there is little prior knowledge to constrain choice of parameters). While raw instability due to multicollinearity should decline with graph size, improper specification will still result in biased coefficient estimates so long as an omitted predictor correlates with an included predictor. Caution is advised.

Value

A glm object

WARNING

Estimation of p* models by maximum pseudo-likelihood is now known to be a dangerous practice. Use at your own risk.

Note

In the long run, support will be included for p* models involving arbitrary functions (much like the system used with cugtest and qaptest).

Author(s)

Carter T. Butts (butts@csc.uci.edu)

References


See Also
eval.edgeperturbation
Examples

# Create a graph with expansiveness and popularity effects
in.str <- rnorm(20, 0, 3)
out.str <- rnorm(20, 0, 3)
tie.str <- outer(out.str, in.str, "+")
tie.p <- apply(tie.str, c(1, 2), function(a) {1/(1+exp(-a))})
g <- rgraph(20, tprob = tie.p)

# Fit a model with expansiveness only
p1 <- pstar(g, effects = "outdegree")

# Fit a model with expansiveness and popularity
p2 <- pstar(g, effects = c("outdegree", "indegree"))

# Fit a model with expansiveness, popularity, and mutuality
p3 <- pstar(g, effects = c("outdegree", "indegree", "mutuality"))

# Compare the model AICs
extractAIC(p1)
extractAIC(p2)
extractAIC(p3)

qaptest

Perform Quadratic Assignment Procedure (QAP) Hypothesis Tests for Graph-Level Statistics

Description

qaptest tests an arbitrary graph-level statistic (computed on dat by FUN) against a QAP null hypothesis, via Monte Carlo simulation of likelihood quantiles. Note that fair amount of flexibility is possible regarding QAP tests on functions of such statistics (see an equivalent discussion with respect to CUG null hypothesis tests in Anderson et al. (1999)). See below for more details.

Usage

qaptest(dat, FUN, reps=1000, ...)

Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Though this could in principle be an array containing only one graph, this is rarely if ever sensible in a QAP-test context.

FUN Function to generate the test statistic. FUN must accept dat and the specified g arguments, and should return a real number.

reps Integer indicating the number of draws to use for quantile estimation. Note that, as for all Monte Carlo procedures, convergence is slower for more extreme quantiles. By default, reps=1000.

... Additional arguments to FUN.
Details

The null hypothesis of the QAP test is that the observed graph-level statistic on graphs \( G_1, G_2, \ldots \) was drawn from the distribution of said statistic evaluated (uniformly) on the set of all relabelings of \( G_1, G_2, \ldots \). Pragmatically, this test is performed by repeatedly (randomly) relabeling the input graphs, recalculating the test statistic, and then evaluating the fraction of draws greater than or equal to (and less than or equal to) the observed value. This accumulated fraction approximates the integral of the distribution of the test statistic over the set of unlabeled input graphs.

The \texttt{qaptest} procedure returns a \texttt{qaptest} object containing the estimated likelihood (distribution of the test statistic under the null hypothesis), the observed value of the test statistic on the input data, and the one-tailed p-values (estimated quantiles) associated with said observation. As usual, the (upper tail) null hypothesis is rejected for significance level alpha if \( p>\text{observation} \) is less than alpha (or \( p<\text{observation} \), for the lower tail); if the hypothesis is undirected, then one rejects if either \( p<\text{observation} \) or \( p>\text{observation} \) is less then alpha/2. Standard caveats regarding the use of null hypothesis testing procedures are relevant here: in particular, bear in mind that a significant result does not necessarily imply that the likelihood ratio of the null model and the alternative hypothesis favors the latter.

In interpreting a QAP test, it is important to bear in mind the nature of the QAP null hypothesis. The QAP test should not be interpreted as evaluating underlying structural differences; indeed, QAP is more accurately understood as testing differences induced by a particular vertex labeling controlling for underlying structure. Where there is substantial automorphism in the underlying structures, QAP will tend to give non-significant results. (In fact, it is impossible to obtain a one-tailed significance level in excess of \( \max_{g \in \{G,H\}} \frac{|\text{Aut}(g)|}{|\text{Perm}(g)|} \) when using a QAP test on a bivariate graph statistic \( f(G,H) \), where Aut(g) and Perm(g) are the automorphism and permutation groups on g, respectively. This follows from the fact that all members of Aut(g) will induce the same values of \( f() \).) By turns, significance under QAP does not necessarily imply that the observed structural relationship is unusual relative to what one would expect from typical structures with (for instance) the sizes and densities of the graphs in question. In contexts in which one’s research question implies a particular labeling of vertices (e.g., "within this group of individuals, do friends also tend to give advice to one another"), QAP can be a very useful way of ruling out spurious structural influences (e.g., some respondents tend to indiscriminately nominate many people (without regard to whom), resulting in a structural similarity which has nothing to do with the identities of those involved). Where one’s question does not imply a labeled relationship (e.g., is the shape of this group’s friendship network similar to that of its advice network), the QAP null hypothesis is inappropriate.

Value

An object of class \texttt{qaptest}, containing

- \texttt{testval} \hspace{1cm} The observed value of the test statistic.
- \texttt{dist} \hspace{1cm} A vector containing the Monte Carlo draws.
- \texttt{pgreq} \hspace{1cm} The proportion of draws which were greater than or equal to the observed value.
- \texttt{pleeq} \hspace{1cm} The proportion of draws which were less than or equal to the observed value.

Author(s)

Carter T. Butts \( \langle \text{buttsc@uci.edu} \rangle \)

References


See Also
cugtest

Examples

```r
#Generate three graphs
g<-array(dim=c(3,10,10))
g[1,,]<-rgraph(10)
g[2,,]<-rgraph(10,tprob=g[1,,]*0.8)
g[3,,]<-1; g[3,1,2]<-0  #This is nearly a clique

#Perform qap tests of graph correlation
q.12<-qaptest(g,gcor,g1=1,g2=2)
q.13<-qaptest(g,gcor,g1=1,g2=3)

#Examine the results
summary(q.12)
plot(q.12)
summary(q.13)
plot(q.13)
```

reachability

---

**Find the Reachability Matrix of a Graph**

Description

`reachability` takes a (possibly directed) graph as input, producing the associated reachability matrix.

Usage

```r
reachability(dat, geodist.precomp=NULL)
```

Arguments

- **dat**
  - A graph adjacency matrix (directed or otherwise)
- **geodist.precomp**
  - Optionally, a precomputed `geodist` object
Details

For a digraph $G = (V, E)$ with vertices $i$ and $j$, let $P_{ij}$ represent a directed $ij$ path. Then the graph

$$R = \langle V(G), \{(i, j) : i, j \in V(G), P_{ij} \in G \} \rangle$$

is said to be the reachability graph of $G$, and the adjacency matrix of $R$ is said to be $G$’s reachability matrix. (Note that when $G$ is undirected, we simply take each undirected edge to be bidirectional.) Vertices which are adjacent in the reachability graph are connected by one or more directed paths in the original graph; thus, structural equivalence classes in the reachability graph are synonymous with strongly connected components in the original structure.

Bear in mind that – as with all matters involving connectedness – reachability is strongly related to size and density. Since, for any given density, almost all structures of sufficiently large size are connected, reachability graphs associated with large structures will generally be complete. Measures based on the reachability graph, then, will tend to become degenerate in the large $|V(G)|$ limit (assuming constant positive density).

Value

A reachability matrix

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

geodist

Examples

#Find the reachability matrix for a sparse random graph
q<-rgraph(10,tprob=0.15)
rg<-reachability(q)
g #Compare the two structures
rg

#Compare to the output of geodist
all(rg==(geodist(q)$counts>0))
Read (N)eo-(O)rg(S)tat Input Files

Description
Reads an input file in NOS format, returning the result as a graph stack.

Usage
read.nos(file)

Arguments
file The file to be imported

Details
NOS format consists of three header lines, followed by a whitespace delimited stack of raw adjacency matrices; the format is not particularly elegant, but turns up in certain legacy applications (mostly at CMU). read.nos provides a quick and dirty way of reading in these files, without the headache of messing with read.table settings.

The content of the NOS format is as follows:
<m>
<n> <o>
<kr1> <kr2> ... <krn> <kc1> <kc2> ... <kcn>
<a111> <a112> ... <a11o>
<a121> <a122> ... <a12o>
...
<a1n1> <a1n2> ... <a1no>
<a211> <a212> ... <a21o>
...
<a2n1> <a2n2> ... <a2no>
...
<amn1> <amn2> ... <amno>
where <abcd> is understood to be the value of the c->d edge in the bth graph of the file. (As one might expect, m, n, and o are the numbers of graphs (matrices), rows, and columns for the data, respectively.) The "k" line contains a list of row and column "colors", categorical variables associated with each row and column, respectively. Although originally intended to communicate exchangability information, these can be used for other purposes (though there are easier ways to deal with attribute data these days).

Value
The imported graph stack

Note
read.nos currently ignores the coloring information.
rgnm

Author(s)
Carter T. Butts (butsc@uci.edu)

See Also
scan, read.table

Examples

rgnm

Draw Density-Conditioned Random Graphs

Description
rgnm generates random draws from a density-conditioned uniform random graph distribution.

Usage
rgnm(n, nv, m, mode = "digraph", diag = FALSE)

Arguments
n
the number of graphs to generate.

nv
the size of the vertex set (\(|V(G)|\)) for the random graphs.

m
the number of edges on which to condition.

mode
"digraph" for directed graphs, or "graph" for undirected graphs.

diag
boolean; should loops be allowed?

Details
rgnm returns draws from the density-conditioned uniform random graph first popularized by the famous work of Erdös and Rényi (the \(G(N, M)\) process). In particular, the pmf of a \(G(N, M)\) process is given by

\[
p(G = g | N, M) = \left( \frac{E_m}{M} \right)^{-1}
\]

where \(E_m\) is the maximum number of edges in the graph. \(E_m\) is equal to \(n v^*(n v - \text{diag}) / (1 + (\text{mode}="\text{graph}"))\). The \(G(N, M)\) process is one of several process which are used as baseline models of social structure. Other well-known baseline models include the Bernoulli graph (the \(G(N, p)\) model of Erdös and Rényi) and the \(\text{U}\mid\text{MAN}\) model of dyadic independence. These are implemented within \textsc{sna} as \texttt{rgraph} and \texttt{rgnm}, respectively.

Value
A matrix or array containing the drawn adjacency matrices
rgraph

Generate Bernoulli Random Graphs

Description

rgraph generates random draws from a Bernoulli graph distribution, with various parameters for controlling the nature of the data so generated.

Usage

rgraph(n, m=1, tprob=0.5, mode="digraph", diag=FALSE, replace=FALSE, tielist=NULL)

Arguments

n The size of the vertex set (|V(G)|) for the random graphs
m The number of graphs to generate
tprob Information regarding tie (edge) probabilities; see below
mode “digraph” for directed data, “graph” for undirected data
diag Should the diagonal entries (loops) be set to zero?
replace Sample with or without replacement from a tie list (ignored if tielist==NULL)
tielist A vector of edge values, from which the new graphs should be bootstrapped

Examples

#Draw 5 random graphs of order 10
all(gden(rgnm(5,10,9,mode="graph"))==0.2) #Density 0.2
all(gden(rgnm(5,10,9))==0.1) #Density 0.1

#Plot a random graph
gplot(rgnm(1,10,20))
**Details**

`rgraph` is a reasonably versatile routine for generating random network data. The graphs so generated are either Bernoulli graphs (graphs in which each edge is a Bernoulli trial, independent conditional on the Bernoulli parameters), or are bootstrapped from a user-provided edge distribution (very handy for CUG tests). In the latter case, edge data should be provided using the `tielist` argument; the exact form taken by the data is irrelevant, so long as it can be coerced to a vector. In the former case, Bernoulli graph probabilities are set by the `tprob` argument as follows:

1. If `tprob` contains a single number, this number is used as the probability of all edges.
2. If `tprob` contains a vector, each entry is assumed to correspond to a separate graph (in order). Thus, each entry is used as the probability of all edges within its corresponding graph.
3. If `tprob` contains a matrix, then each entry is assumed to correspond to a separate edge. Thus, each entry is used as the probability of its associated edge in each graph which is generated.
4. Finally, if `tprob` contains a three-dimensional array, then each entry is assumed to correspond to a particular edge in a particular graph, and is used as the associated probability parameter.

Finally, note that `rgraph` will symmetrize all generated networks if `mode` is set to “graph” by copying down the upper triangle. The lower half of `tprob`, where applicable, must still be specified, however.

**Value**

A graph stack

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**References**


**See Also**

`rmperm`, `rgnm`, `rguman`

**Examples**

```r
g <- rgraph(10, 3, tprob = c(0.1, 0.9, 0.5))
g.p <- matrix(runif(25, 0, 1), nrow = 5)
g <- rgraph(5, 2, tprob = g.p)
```

# Generate three graphs with different densities

# Generate from a matrix of Bernoulli parameters
rguman generates random draws from a dyad census-conditioned uniform random graph distribution.

**Usage**

```r
guman(n, nv, mut = 0.25, asym = 0.5, null = 0.25, method = c("probability", "exact"))
```

**Arguments**

- `n`: the number of graphs to generate.
- `nv`: the size of the vertex set (`|V(G)|`) for the random graphs.
- `mut`: if `method=="probability"`, the probability of obtaining a mutual dyad; otherwise, the number of mutual dyads.
- `asym`: if `method=="probability"`, the probability of obtaining an asymmetric dyad; otherwise, the number of asymmetric dyads.
- `null`: if `method=="probability"`, the probability of obtaining a null dyad; otherwise, the number of null dyads.
- `method`: the generation method to use. "probability" results in a multinomial dyad distribution (conditional on the underlying rates), while "exact" results in a uniform draw conditional on the exact dyad distribution.

**Details**

A simple generalization of the Erdős–Rényi family, the U|MAN distributions are uniform on the set of graphs, conditional on order (size) and the dyad census. As with the E-R case, there are two U|MAN variants. The first (corresponding to `method=="probability"`) takes dyad states as independent multinomials with parameters `m` (for mutuals), `a` (for asymmetrics), and `n` (for nulls). The resulting pmf is then

\[
p(G = g|m, a, n) = \frac{(M + A + N)!}{M!A!N!} m^M a^A n^N,
\]

where `M`, `A`, and `N` are realized counts of mutual, asymmetric, and null dyads, respectively. (See `dyad.census` for an explication of dyad types.)

The second U|MAN variant is selected by `method=="exact"`, and places equal mass on all graphs having the specified (exact) dyad census. The corresponding pmf is

\[
p(G = g|M, A, N) = \frac{M!A!N!}{(M + A + N)!}.
\]

U|MAN graphs provide a natural baseline model for networks which are constrained by size, density, and reciprocity. In this way, they provide a bridge between edgewise models (e.g., the E-R family) and models with higher order dependence (e.g., the Markov graphs).
rgws

Value
A matrix or array containing the drawn adjacency matrices

Author(s)
Carter T. Butts (butts@uci.edu)

References

See Also
rgraph, rgnm, dyad.census

Examples
# Show some examples of extreme U|M plot graphs
> gplot(rguman(1,10,mut=45,asym=0,null=0,method="exact")) # Clique
> gplot(rguman(1,10,mut=0,asym=45,null=0,method="exact")) # Tournament
> gplot(rguman(1,10,mut=0,asym=0,null=45,method="exact")) # Empty

# Draw a sample of multinomial U|M plot graphs
> g<-rguman(5,10,mut=0.15,asym=0.05,null=0.8)

# Examine the dyad census
> dyad.census(g)

rgws

Draw From the Watts-Strogatz Rewiring Model

Description
rgws generates draws from the Watts-Strogatz rewired lattice model. Given a set of input graphs, rewire.ws performs a (dyadic) rewiring of those graphs.

Usage
rgws(n, nv, d, z, p)
rewire.ud(g, p)
rewire.ws(g, p)

Arguments
n the number of draws to take.
nv the number of vertices per lattice dimension.
d the dimensionality of the underlying lattice.
z the nearest-neighbor threshold for local ties.
p the dyadic rewiring probability.
g a graph or graph stack.
Details

A Watts-Strogatz graph process generates a random graph via the following procedure. First, a \(d\)-dimensional uniform lattice is generated, here with \(nv\) vertices per dimension (i.e., \(nv^d\) vertices total). Next, all \(z\) neighbors are connected, based on geodesics of the underlying lattice. Finally, each non-null dyad in the resulting augmented lattice is "rewired" with probability \(p\), where the rewiring operation exchanges the initial dyad state with the state of a uniformly selected null dyad sharing exactly one endpoint with the original dyad. (In the standard case, this is equivalent to choosing an endpoint of the dyad at random, and then transferring the dyadic edges to/from that endpoint to another randomly chosen vertex. Hence the "rewiring" metaphor.) For \(p==0\), the W-S process generates (deterministic) uniform lattices, approximating a uniform G(N,M) process as \(p\) approaches 1. Thus, \(p\) can be used to tune overall entropy of the process. A well-known property of the W-S process is that (for large \(nv^d\) and small \(p\)) it generates draws with short expected mean geodesic distances (approaching those found in uniform graphs) while maintaining high levels of local "clustering" (i.e., transitivity). It has thus been proposed as one potential mechanism for obtaining "small world" structures.

\texttt{rgws} produces independent draws from the above process, returning them as an adjacency matrix (if \(n==1\)) or array (otherwise). \texttt{rewire.ws}, on the other hand, applies the rewiring phase of the W-S process to one or more input graphs. This can be used to explore local perturbations of the original graphs, conditioning on the dyad census. \texttt{rewire.ud} is similar to \texttt{rewire.ws}, save in that all dyads are eligible for rewiring (not just non-null dyads), and exchanges with non-null dyads are permitted. This process may be easier to work with than standard W-S rewiring in some cases.

Value

A graph or graph stack containing draws from the appropriate W-S process.

Warning

Remember that the total number of vertices in the graph is \(nv^d\). This can get out of hand very quickly.

Note

\texttt{rgws} generates non-toroidal lattices; some published work in this area utilizes underlying toroids, so users should check for this prior to comparing simulations against published results.

Author(s)

Carter T. Butts \(\langle\text{buttsc@uci.edu}\rangle\)

References


See Also

\texttt{rgnm}, \texttt{rgraph}
Examples

# Generate Watts-Strogatz graphs, w/increasing levels of rewiring

gplot(rgws(1,100,1,2,0))  # No rewiring

gplot(rgws(1,100,1,2,0.01))  # 1

gplot(rgws(1,100,1,2,0.05))  # 5

gplot(rgws(1,100,1,2,0.1))   # 10

gplot(rgws(1,100,1,2,1))     # 100

# Start with a simple graph, then rewire it

g <- matrix(0, 50, 50)
g[1,] <- 1; g[,1] <- 1  # Create a star

plot(g)

plot(rewire.ws(g, 0.05))  # 5

rmperm

Randomly Permute the Rows and Columns of an Input Matrix

Description

Given an input matrix (or stack thereof), rmperm performs a (random) simultaneous row/column permutation of the input data.

Usage

rperm(m)

Arguments

m A matrix, or stack thereof

Details

Random matrix permutations are the essence of the QAP test; see qaptest for details.

Value

The permuted matrix (or matrices)

Author(s)

Carter T. Butts (butts@uci.edu)

See Also

rperm
Examples

# Generate an input matrix
g <- rgraph(5)
g # Examine it

# Examine a random permutation
rmperm(g)

rperm

Draw a Random Permutation Vector with Exchangeability Constraints

Description

Draws a random permutation on 1:length(exchange.list) such that no two elements whose corresponding exchange.list values are different are interchanged.

Usage

rperm(exchange.list)

Arguments

exchange.list

A vector such that the permutation vector may exchange the ith and jth positions iff exchange.list[i] == exchange.list[j]

Details

rperm draws random permutation vectors given the constraints of exchangeability described above. Thus, rperm(c(0, 0, 0, 0)) returns a random permutation of four elements in which all exchanges are allowed, while rperm(c(1, 1, "a", "a")) (or similar) returns a random permutation of four elements in which only the first/second and third/fourth elements may be exchanged. This turns out to be quite useful for searching permutation spaces with exchangeability constraints (e.g., for structural distance estimation).

Value

A random permutation vector satisfying the given constraints

Author(s)

Carter T. Butts (buttsc@uci.edu)

See Also

rmperm

Examples

rperm(c(0, 0, 0, 0)) # All elements may be exchanged
rperm(c(0, 0, 0, 1)) # Fix the fourth element
rperm(c(0, 0, 1, 1)) # Allow {1, 2} and {3, 4} to be swapped
rperm(c("a", 4, "x", 2)) # Fix all elements (the identity permutation)
sdmat

Estimate the Structural Distance Matrix for a Graph Set

Description
Estimates the structural distances among all elements of dat using the method specified in method.

Usage
sdmat(dat, normalize=FALSE, diag=FALSE, mode="digraph", output="matrix", method="mc", exchange.list=rep(0, dim(dat)[2]), ...)

Arguments
dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. This data need not be dichotomous, and missing values are allowed.
normalize Divide by the number of available dyads?
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.
output "matrix" for matrix output, "dist" for a dist object.
method Method to be used to search the space of accessible permutations; must be one of "none", "exhaustive", "anneal", "hillclimb", or "mc".
exchange.list Information on which vertices are exchangeable (see below); this must be a single number, a vector of length n, or a nx2 matrix.
... Additional arguments to lab.optimize

Details
The structural distance between two graphs G and H is defined as

\[ d_S (G, H | L_G, L_H) = \min_{L_G, L_H} d(\ell (G), \ell (H)) \]

where \( L_G \) is the set of accessible permutations/labelings of G, and \( \ell (G) \) is a permutation/relabeling of the vertices of G (\( \ell (G) \in L_G \)). The set of accessible permutations on a given graph is determined by the theoretical exchangeability of its vertices; in a nutshell, two vertices are considered to be theoretically exchangeable for a given problem if all predictions under the conditioning theory are invariant to a relabeling of the vertices in question (see Butts and Carley (2001) for a more formal exposition). Where no vertices are exchangeable, the structural distance becomes the its labeled counterpart (here, the Hamming distance). Where all vertices are exchangeable, the structural distance reflects the distance between unlabeled graphs; other cases correspond to distance under partial labeling.

The accessible permutation set is determined by the exchange.list argument, which is dealt with in the following manner. First, exchange.list is expanded to fill an nx2 matrix. If
exchange.list is a single number, this is trivially accomplished by replication; if exchange.list is a vector of length n, the matrix is formed by cbinding two copies together. If exchange.list is already an nx2 matrix, it is left as-is. Once the nx2 exchangeability matrix has been formed, it is interpreted as follows: columns refer to graphs 1 and 2, respectively; rows refer to their corresponding vertices in the original adjacency matrices; and vertices are taken to be theoretically exchangeable iff their corresponding exchangeability matrix values are identical. To obtain an unlabeled distance (the default), then, one could simply let exchange.list equal any single number. To obtain the Hamming distance, one would use the vector 1:n.

Because the set of accessible permutations is, in general, very large \((o(n!))\), searching the set for the minimum distance is a non-trivial affair. Currently supported methods for estimating the structural distance are hill climbing, simulated annealing, blind monte carlo search, or exhaustive search (it is also possible to turn off searching entirely). Exhaustive search is not recommended for graphs larger than size 8 or so, and even this may take days; still, this is a valid alternative for small graphs. Blind monte carlo search and hill climbing tend to be suboptimal for this problem and are not, in general recommended, but they are available if desired. The preferred (and default) option for permutation search is simulated annealing, which seems to work well on this problem (though some tinkering with the annealing parameters may be needed in order to get optimal performance). See the help for \texttt{lab.optimize} for more information regarding these options.

Structural distance matrices may be used in the same manner as any other distance matrices (e.g., with multidimensional scaling, cluster analysis, etc.) Classical null hypothesis tests should not be employed with structural distances, and QAP tests are almost never appropriate (save in the uniquely labeled case). See \texttt{cugtest} for a more reasonable alternative.

Value

A matrix of distances (or an object of class \texttt{dist})

Warning

The search process can be \textit{very slow}, particularly for large graphs. In particular, the \texttt{exhaustive} method is order factorial, and will take approximately forever for unlabeled graphs of size greater than about 7-9.

Note

For most applications, \texttt{sdmat} is dominated by \texttt{structdist}; the former is retained largely for reasons of compatibility.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

\texttt{hdist,structdist}
Examples

# Generate two random graphs
g <- array(dim = c(3, 5, 5))
g[1, , ] <- rgraph(5)
g[2, , ] <- rgraph(5)

# Copy one of the graphs and permute it
g[3, , ] <- rmperm(g[2, , ])

# What are the structural distances between the labeled graphs?
sdm(g, exchange.list = 1:5)

# What are the structural distances between the underlying unlabeled graphs?
sdm(g, method = "anneal", prob.init = 0.9, prob.decay = 0.85,
   freeze.time = 50, full.neighborhood = TRUE)

sedist Find a Matrix of Distances Between Positions Based on Structural Equivalence

Description

sedist uses the graphs indicated by g in dat to assess the extent to which each vertex is structurally equivalent; joint.analysis determines whether this analysis is simultaneous, and method determines the measure of approximate equivalence which is used.

Usage

sedist(dat, g = c(1:dim(dat)[1]), method = "hamming",
   joint.analysis = FALSE, mode = "digraph", diag = FALSE, code.diss = FALSE)

Arguments

dat A graph or graph stack
g A vector indicating which elements of dat should be examined
method One of "correlation", "euclidean", "hamming", and "gamma"
joint.analysis Should equivalence be assessed across all networks jointly (TRUE), or individually within each (FALSE)
mode "digraph" for directed data, otherwise "graph"
diag Boolean indicating whether diagonal entries (loops) should be treated as meaningful data
code.diss Reverse-code the raw comparison values
Details

sedist provides a basic tool for assessing the (approximate) structural equivalence of actors. (Two vertices i and j are said to be structurally equivalent if i->k iff j->k for all k.) SE similarity/difference scores are computed by comparing vertex rows and columns using the measure indicated by method:

1. correlation: the product-moment correlation
2. euclidean: the euclidean distance
3. hamming: the Hamming distance
4. gamma: the gamma correlation

Once these similarities/differences are calculated, the results can be used with a clustering routine (such as `equiv.clust`) or an MDS (such as `cmdscale`).

Value

A matrix of similarity/difference scores

Note

Be careful to verify that you have computed what you meant to compute, with respect to similarities/differences. Also, note that (despite its popularity) the product-moment correlation can give rather strange results in some cases.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


See Also

`equiv.clust`, `blockmodel`

Examples

```r
#Create a random graph with _some_ edge structure
g.p<-sapply(runif(20,0,1),rep,20) #Create a matrix of edge probabilities
g<-rgraph(20,tprob=g.p) #Draw from a Bernoulli graph distribution

#Get SE distances
g.se<-sedist(g)

#Plot a metric MDS of vertex positions in two dimensions
plot(cmdscale(as.dist(g.se)))
```
Description

These functions are provided for compatibility with older versions of \texttt{sna} only, and may be defunct as soon as the next release.

Details

The following \texttt{sna} functions are currently deprecated:

\texttt{addisolates}

The original help pages for these functions can be found at \texttt{help("oldName-deprecated")}. Please avoid using them, since they will disappear.

Author(s)

Carter T. Butts \texttt{(butsc@uci.edu)}

See Also

\texttt{Deprecated}

---

Description

Internal \texttt{sna} functions.

Usage

\begin{verbatim}
bbnam.jntlik(dat, log=FALSE, ...)
bbnam.jntlik.slice(s, dat, a, em, ep, log=FALSE)
bbnam.probtie(dat, i, j, npriorij, em, ep)
\end{verbatim}

Details

These are not to be called by the end user.
Description

These operators allow for algebraic manipulation of graph adjacency matrices.

Usage

\[ x \mathbin{\%c\%} y \]

Arguments

- \( x \) an (unvalued) adjacency matrix.
- \( y \) another (unvalued) adjacency matrix.

Details

Currently, only one operator is supported. \( x \mathbin{\%c\%} y \) returns the adjacency matrix of the composition of graphs with adjacency matrices \( x \) and \( y \) (respectively). (Note that this may contain loops.)

Value

The resulting adjacency matrix.

Author(s)

Carter T. Butts (buttsc@uci.edu)

References


Examples

```r
#Create an in-star
g <- matrix(0, 6, 6)
g[2:6, 1] <- 1
gplot(g)

#Compose g with its transpose
gc <- g %c% t(g)
gplot(gc, diag=TRUE)
gc
```
sr2css  

*Convert a Row-wise Self-Report Matrix to a CSS Matrix with Missing Observations*

**Description**

Given a matrix in which the ith row corresponds to i’s reported relations, sr2css creates a graph stack in which each element represents a CSS slice with missing observations.

**Usage**

```
sr2css(net)
```

**Arguments**

| net       | An adjacency matrix |

**Details**

A cognitive social structure (CSS) is an nxnxn array in which the ith matrix corresponds to the ith actor’s perception of the entire network. Here, we take a conventional self-report data structure and put it in CSS format for routines (such as `bbnam`) which require this.

**Value**

An array (graph stack) containing the CSS

**Note**

A row-wise self-report matrix doesn’t contain a great deal of data, and the data in question is certainly not an ignorable sample of the individual’s CSS for most purposes. The provision of this routine should not be perceived as license to substitute SR for CSS data at will.

**Author(s)**

Carter T. Butts (buttc@uci.edu)

**References**


**Examples**

```r
# Start with some random reports
g<rgraph(10)

# Transform to CSS format
c<-sr2css(g)
```
stackcount

Description
Returns the number of graphs in the stack provided by \( d \).

Usage
\[
\text{stackcount}(d)
\]

Arguments
- \( d \) A graph or graph stack

Details

Value
The number of graphs in \( d \)

Author(s)
Carter T. Butts (buttsc@uci.edu)

See Also
nties

Examples
\[
\text{stackcount}(\text{rgraph}(4,8)) == 8
\]

stresscent

Description
stresscent takes a graph stack (dat) and returns the stress centralities of positions within one graph (indicated by nodes and \( g \), respectively). Depending on the specified mode, stress on directed or undirected geodesics will be returned; this function is compatible with \texttt{centralization}, and will return the theoretical maximum absolute deviation (from maximum) conditional on size (which is used by \texttt{centralization} to normalize the observed centralization score).

Usage
\[
\text{stresscent}(\text{dat}, g=1, \text{nodes}=c(1:dim(\text{dat})[2]), \text{gmode}="\text{digraph}"), \text{diag}=\text{FALSE}, \text{tmaxdev}=\text{FALSE}, \text{cmode}="\text{directed}", \text{geodist.precomp}=\text{NULL}, \text{rescale}=\text{FALSE})
\]
Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. Alternately, this can be an n x n matrix (if only one graph is involved).

g Integer indicating the index of the graph for which centralities are to be calculated. By default, g=1.

nodes List indicating which nodes are to be included in the calculation. By default, all nodes are included.

gmode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. gmode is set to "digraph" by default.

diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.

tmaxdev Boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, tmaxdev==FALSE.

cmode String indicating the type of betweenness centrality being computed (directed or undirected geodesics).

geodist.precomp A geodist object precomputed for the graph to be analyzed (optional)

rescale If true, centrality scores are rescaled such that they sum to 1.

Details

The stress of a vertex, v, is given by

\[ C_S(v) = \sum_{i,j \neq j,i \neq v,j \neq v} g_{ijk} \]

where \( g_{ijk} \) is the number of geodesics from i to k through j. Conceptually, high-stress vertices lie on a large number of shortest paths between other vertices; they can thus be thought of as “bridges” or “boundary spanners.” Compare this with betweenness, which weights shortest paths by the inverse of their redundancy.

Value

A vector of centrality scores

Note

Judicious use of geodist.precomp can save a great deal of time when computing multiple path-based indices on the same network.

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

References

structdist

See Also
centralization

Examples

g<-rgraph(10)  #Draw a random graph with 10 members
stresscent(g)  #Compute stress scores

structdist

Find the Structural Distances Between Two or More Graphs

Description

structdist returns the structural distance between the labeled graphs g1 and g2 in stack dat based on Hamming distance for dichotomous data, or else the absolute (manhattan) distance. If normalize is true, this distance is divided by its dichotomous theoretical maximum (conditional on |V(G)|).

Usage

structdist(dat, g1=c(1:dim(dat)[1]), g2=c(1:dim(dat)[1]), normalize=FALSE, diag=FALSE, mode="digraph", method="anneal", reps=1000, prob.init=0.9, prob.decay=0.85, freeze.time=25, full.neighborhood=TRUE, mut=0.05, pop=20, trials=5, exchange.list=rep(0, dim(dat)[2]))

Arguments

dat Data array to be analyzed. By assumption, the first dimension of the array indexes the graph, with the next two indexing the actors. This data need not be dichotomous, and missing values are allowed.
g1 A vector indicating which graphs to compare (by default, all elements of dat)
g2 A vector indicating against which the graphs of g1 should be compared (by default, all graphs)
normalize Divide by the number of available dyads?
diag Boolean indicating whether or not the diagonal should be treated as valid data. Set this true if and only if the data can contain loops. diag is FALSE by default.
mode String indicating the type of graph being evaluated. "digraph" indicates that edges should be interpreted as directed; "graph" indicates that edges are undirected. mode is set to "digraph" by default.
method Method to be used to search the space of accessible permutations; must be one of ?none?, ?exhaustive?, ?anneal?, ?hillclimb?, or ?mc?.
reps Number of iterations for monte carlo method.
prob.init Initial acceptance probability for the annealing routine.
prob.decay Cooling multiplier for the annealing routine.
freeze.time Freeze time for the annealing routine.
full.neighborhood
Should the annealer evaluate the full neighborhood of pair exchanges at each iteration?

mut
GA Mutation rate (currently ignored)

pop
GA population (currently ignored)

trials
Number of GA populations (currently ignored)

exchange.list
Information on which vertices are exchangeable (see below); this must be a single number, a vector of length n, or a nx2 matrix.

Details
The structural distance between two graphs G and H is defined as

\[ d_S(G, H | L_G, L_H) = \min_{L_G, L_H} d(\ell(G), \ell(H)) \]

where \( L_G \) is the set of accessible permutations/labelings of G, and \( \ell(G) \) is a permutation/relabeling of the vertices of G (\( \ell(G) \in L_G \)). The set of accessible permutations on a given graph is determined by the theoretical exchangeability of its vertices; in a nutshell, two vertices are considered to be theoretically exchangeable for a given problem if all predictions under the conditioning theory are invariant to a relabeling of the vertices in question (see Butts and Carley (2001) for a more formal exposition). Where no vertices are exchangeable, the structural distance becomes its labeled counterpart (here, the Hamming distance). Where all vertices are exchangeable, the structural distance reflects the distance between unlabeled graphs; other cases correspond to distance under partial labeling.

The accessible permutation set is determined by the exchange.list argument, which is dealt with in the following manner. First, exchange.list is expanded to fill an nx2 matrix. If exchange.list is a single number, this is trivially accomplished by replication; if exchange.list is a vector of length n, the matrix is formed by cbinding two copies together. If exchange.list is already an nx2 matrix, it is left as-is. Once the nx2 exchangeability matrix has been formed, it is interpreted as follows: columns refer to graphs 1 and 2, respectively; rows refer to their corresponding vertices in the original adjacency matrices; and vertices are taken to be theoretically exchangeable if their corresponding exchangeability matrix values are identical. To obtain an unla-
beled distance (the default), then, one could simply let exchange.list equal any single number. To obtain the Hamming distance, one would use the vector 1:n.

Because the set of accessible permutations is, in general, very large (\( o(n!) \)), searching the set for the minimum distance is a non-trivial affair. Currently supported methods for estimating the structural distance are hill climbing, simulated annealing, blind monte carlo search, or exhaustive search (it is also possible to turn off searching entirely). Exhaustive search is not recommended for graphs larger than size 8 or so, and even this may take days; still, this is a valid alternative for small graphs. Blind monte carlo search and hill climbing tend to be suboptimal for this problem and are not, in general recommended, but they are available if desired. The preferred (and default) option for permutation search is simulated annealing, which seems to work well on this problem (though some tinkering with the annealing parameters may be needed in order to get optimal performance). See the help for lab.optimize for more information regarding these options.

Structural distance matrices may be used in the same manner as any other distance matrices (e.g., with multidimensional scaling, cluster analysis, etc.) Classical null hypothesis tests should not be employed with structural distances, and QAP tests are almost never appropriate (save in the uniquely labeled case). See cugtest for a more reasonable alternative.
Value

A structural distance matrix

Warning

The search process can be very slow, particularly for large graphs. In particular, the exhaustive method is order factorial, and will take approximately forever for unlabeled graphs of size greater than about 7-9.

Note

Consult Butts and Carley (2001) for advice and examples on theoretical exchangeability.

Author(s)

Carter T. Butts (butts@uci.edu)

References


See Also

hdist, sdmat

Examples

```r
#Generate two random graphs
g<-array(dim=c(3,5,5))
g[1,,]<-rgraph(5)
g[2,,]<-rgraph(5)
#Copy one of the graphs and permute it
g[3,,]<-rmperm(g[2,,])
#What are the structural distances between the labeled graphs?
structdist(g,exchange.list=1:5)
#What are the structural distances between the underlying unlabeled graphs?
structdist(g,method="anneal", prob.init=0.9, prob.decay=0.85, freeze.time=50, full.neighborhood=TRUE)
```

---

**summary.bayes.factor**

*Detailed Summaries of Bayes Factor Objects*

**Description**

Returns a `bayes.factor` summary object.
**summary.bbnam**

**Usage**

```
summary.bayes.factor(object, ...)
```

**Arguments**

- `object` An object of class `bayes.factor`
- `...` Further arguments passed to or from other methods

**Value**

An object of class `summary.bayes.factor`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**See Also**

`bbnam.bf`

**Examples**

---

**summary.bbnam** *Detailed Summaries of bbnam Objects*

---

**Description**

Returns a `bbnam` summary object

**Usage**

```
summary.bbnam(object, ...)
```

**Arguments**

- `object` An object of class `bbnam`
- `...` Further arguments passed to or from other methods

**Value**

An object of class `summary.bbnam`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**See Also**

`bbnam`
**summary.blockmodel**  
*Detailed Summaries of blockmodel Objects*

**Description**  
Returns a blockmodel summary object.

**Usage**  
`summary.blockmodel(object, ...)`

**Arguments**  
- `object`: An object of class `blockmodel`
- `...`: Further arguments passed to or from other methods

**Value**  
An object of class `summary.blockmodel`

**Author(s)**  
Carter T. Butts (buttsc@uci.edu)

**See Also**  
`blockmodel`

**Examples**

**summary.cugtest**  
*Detailed Summaries of cugtest Objects*

**Description**  
Returns a cugtest summary object

**Usage**  
`summary.cugtest(object, ...)`

**Arguments**  
- `object`: An object of class `cugtest`
- `...`: Further arguments passed to or from other methods
**summary.lnam**

**Value**

An object of class `summary.cugtest`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**See Also**

`cugtest`

**Examples**

```r
# Example usage of summary.lnam

# Generate some example lnam object
obj <- lnam()

# Summarize the object
summary_obj <- summary.lnam(obj)

# Details of the summary object
summary_obj
```

**Description**

Returns a `lnam` summary object.

**Usage**

```r
summary.lnam(object, ...)
```

**Arguments**

- `object` an object of class `lnam`
- `...` additional arguments.

**Value**

An object of class `summary.lnam`.

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**See Also**

`lnam`
**summary.netcancor**  
*Detailed Summaries of netcancor Objects*

### Description

Returns a netcancor summary object

### Usage

```
summary.netcancor(object, ...)
```

### Arguments

- **object**
  - An object of class netcancor
- **...**
  - Further arguments passed to or from other methods

### Value

An object of class `summary.netcancor`

### Author(s)

Carter T. Butts  
buttsc@uci.edu

### See Also

`netcancor`

### Examples

---

**summary.netlm**  
*Detailed Summaries of netlm Objects*

### Description

Returns a netlm summary object

### Usage

```
summary.netlm(object, ...)
```

### Arguments

- **object**
  - An object of class netlm
- **...**
  - Further arguments passed to or from other methods
Value

An object of class summary.netlm

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

See Also

netlm

Examples

summary.netlogit  Detailed Summaries of netlogit Objects

Description

Returns a netlogit summary object

Usage

summary.netlogit(object, ...)

Arguments

object  An object of class netlogit
...  Further arguments passed to or from other methods

Value

An object of class summary.netlogit

Author(s)

Carter T. Butts ⟨buttsc@uci.edu⟩

See Also

netlogit

Examples
**summary.qaptest**  
*Detailed Summaries of qaptest Objects*

**Description**

Returns a `qaptest` summary object

**Usage**

```r
summary.qaptest(object, ...)
```

**Arguments**

- `object`  
  An object of class `qaptest`
- `...`  
  Further arguments passed to or from other methods

**Value**

An object of class `summary.qaptest`

**Author(s)**

Carter T. Butts (buttsc@uci.edu)

**See Also**

`qaptest`

**Examples**

---

**symmetrize**  
*Symmetrize an Adjacency Matrix*

**Description**

Symmetrizes the elements of `mats` according to the rule in `rule`.

**Usage**

```r
symmetrize(mats, rule="weak")
```

**Arguments**

- `mats`  
  A graph or graph stack
- `rule`  
  One of “upper”, “lower”, “strong” or “weak”
Details

The rules used by `symmetrize` are as follows:

1. upper: Copy the upper triangle over the lower triangle
2. lower: Copy the lower triangle over the upper triangle
3. strong: i<->j iff i->j and i<-j (AND rule)
4. weak: i<->j iff i->j or i<-j (OR rule)

Value

The symmetrized graph stack

Author(s)

Carter T. Butts (butts@uci.edu)

References


Examples

```r
#Generate a graph
g<-rgraph(5)

#Weak symmetrization
symmetrize(g)

#Strong symmetrization
symmetrize(g,rule="strong")
```

triad.census

`triad.census` returns the Davis and Leinhardt triad census of the elements of `dat` indicated by `g`.

Usage

`triad.census(dat, g=l:stackcount(dat))`

Arguments

- `dat`: A graph or graph stack
- `g`: The elements of `dat` to process
**Details**

The Davis and Leinhardt triad census consists of a classification of all triads into one of 16 different categories; the resulting distribution can be compared against various null models to test for the presence of configural biases (e.g., transitivity bias). triad.census is a front end for the triad.classify routine, performing the classification for all triads within the selected graphs. The results are placed in the order indicated by the column names; this is the same order as presented in the triad.classify documentation, to which the reader is referred for additional details.

Compare triad.census to dyad.census, the dyadic equivalent.

**Value**

A matrix whose 16 columns contain the counts of triads by class for each graph

**Warning**

Valued data may cause strange behavior with this routine. Dichotomize the data first.

**Author(s)**

Carter T. Butts ⟨buttsc@uci.edu⟩

**References**


**See Also**

triad.classify, dyad.census, gtrans

**Examples**

```r
#Generate a triad census of random data with varying densities
triad.census(rgraph(15,5,tprob=c(0.1,0.25,0.5,0.75,0.9)))
```

---

**triad.classify**

*Compute the Davis and Leinhardt Classification of a Given Triad*

**Description**

triad.classify returns the Davis and Leinhardt classification of the triad indicated by tri in the gth graph of stack dat.

**Usage**

`triad.classify(dat, g=1, tri=c(1, 2, 3))`
Arguments

- **dat**: A graph or graph stack
- **g**: The index of the graph to be analyzed
- **tri**: A triple containing the indices of the triad to be classified

Details

Every unoriented directed triad may occupy one of 16 distinct states. These states were used by Davis and Leinhardt as a basis for classifying triads within a larger structure; the distribution of triads within a graph (see `triad.census`), for instance, is linked to a range of substantive hypotheses (e.g., concerning structural balance). The Davis and Leinhardt classification scheme describes each triad by a string of four elements: the number of mutual (complete) dyads within the triad; the number of asymmetric dyads within the triad; the number of null (empty) dyads within the triad; and a configuration code for the triads which are not uniquely distinguished by the first three distinctions. The complete list of classes is as follows.

- **003**: \(a \not\leftrightarrow b \not\leftrightarrow c, a \not\leftrightarrow c\)
- **012**: \(a \rightarrow b \not\leftrightarrow c, a \not\leftrightarrow c\)
- **102**: \(a \leftrightarrow b \not\leftrightarrow c, a \not\leftrightarrow c\)
- **021D**: \(a \leftrightarrow b \rightarrow c, a \not\leftrightarrow c\)
- **021U**: \(a \rightarrow b \leftrightarrow c, a \not\leftrightarrow c\)
- **021C**: \(a \rightarrow b \rightarrow c, a \not\leftrightarrow c\)
- **111D**: \(a \not\leftrightarrow b \rightarrow c, a \leftrightarrow c\)
- **111U**: \(a \not\leftrightarrow b \leftrightarrow c, a \leftrightarrow c\)
- **111C**: \(a \not\leftrightarrow b \rightarrow c, a \leftrightarrow c\)
- **030T**: \(a \leftrightarrow b \leftarrow c, a \rightarrow c\)
- **030C**: \(a \leftrightarrow b \leftrightarrow c, a \rightarrow c\)
- **201**: \(a \leftrightarrow b \not\leftrightarrow c, a \leftrightarrow c\)
- **120D**: \(a \leftrightarrow b \rightarrow c, a \leftrightarrow c\)
- **120U**: \(a \rightarrow b \leftrightarrow c, a \leftrightarrow c\)
- **120C**: \(a \rightarrow b \rightarrow c, a \leftrightarrow c\)
- **210**: \(a \rightarrow b \leftrightarrow c, a \leftrightarrow c\)
- **300**: \(a \leftrightarrow b \leftrightarrow c, a \leftrightarrow c\)

These codes are returned by `triad.classify` as strings.

Value

A string containing the triad classification, or NA if one or more edges were missing

Warning

Valued data and/or loops may cause strange behavior with this routine. Dichotomize/remove loops first.

Author(s)

Carter T. Butts (butts@uci.edu)
References


See Also

triad.census, gtrans

Examples

# Generate a random graph
g<-rgraph(10)

# Classify the triads (1,2,3) and (2,3,4)
triad.classify(g,tri=c(1,2,3))
triad.classify(g,tri=c(1,2,3))

# Plot the triads in question
gplot(g[1:3,1:3])
gplot(g[2:4,2:4])

upper.tri.remove  Remove the Upper Triangles of Adjacency Matrices in a Graph Stack

Description

Returns the input graph stack, with the upper triangle entries removed/replaced as indicated.

Usage

upper.tri.remove(dat, remove.val=NA)

Arguments

dat  A graph stack
remove.val  The value with which to replace the existing upper triangles

Details

upper.tri.remove is simply a convenient way to apply g[upper.tri(g)]<-remove.val to an entire stack of adjacency matrices at once.

Value

The updated graph stack

Author(s)

Carter T. Butts  (buttsc@uci.edu)
upper.tri.remove

See Also

upper.tri, lower.tri.remove, diag.remove

Examples

# Generate a random graph stack
g <- rgraph(3, 5)
# Remove the upper triangles
g <- upper.tri.remove(g)
Index

*Topic algebra
  gtrans, 79
  make.stochastic, 99
  reachability, 137
*Topic aplot
  gplot.arrow, 56
  gplot.loop, 61
  gplot.vertex, 64
  gplot3d.arrow, 67
  gplot3d.loop, 71
*Topic array
  make.stochastic, 99
  numperm, 109
  rmperm, 146
  rperm, 147
  symmetrize, 165
  upper.tri.remove, 169
*Topic classif
  blockmodel, 11
  sedist, 150
*Topic cluster
  equiv.clust, 33
  gclust.centralgraph, 42
  sedist, 150
*Topic distribution
  rgnm, 140
  rgraph, 141
  rguman, 143
  rgws, 144
  rmperm, 146
  rperm, 147
*Topic dplot
  gplot.layout, 57
  gplot3d.layout, 68
*Topic file
  read.nos, 139
*Topic graphs
  betweenness, 9
  closeness, 19
  consensus, 24
  gden, 46
  geodist, 50
  gplot, 53
  gplot.arrow, 56
  gplot.layout, 57
  gplot.loop, 61
  gplot.target, 62
  gplot.vertex, 64
  gplot3d, 65
  gplot3d.arrow, 67
  gplot3d.layout, 68
  gplot3d.loop, 71
  is.connected, 88
  is.isolate, 89
  isolates, 90
  pstar, 132
  rgmn, 140
  rguman, 143
  rgws, 144
  sna_operators, 153
*Topic hplot
  gclust.boxstats, 40
  gdist.plotdiff, 47
  gdist.plotstats, 48
  gplot, 53
  gplot.target, 62
  gplot3d, 65
  plot.bbnam, 110
  plot.blockmodel, 111
  plot.cugtest, 112
  plot.equiv.clust, 113
  plot.lnam, 115
  plot.quaptest, 115
  plot.sociomatrix, 116
*Topic htest
  cugtest, 26
  qaptest, 135
*Topic internal
  sna_internal, 152
*Topic iteration
  gapply, 39
*Topic logic
  is.connected, 88
  is.isolate, 89
*Topic manip
  add.isolates, 1
INDEX

addisolates-deprecated, 2
blockmodel.expand, 13
diag.remove, 30
event2dichot, 38
gapply, 39
gvectorize, 80
interval.graph, 86
lower.tri.remove, 97
make.stochastic, 99
sr2css, 154
symmetrize, 165
upper.tri.remove, 169

*Topic math

add.isolates, 1
addisolates-deprecated, 2
bbnam, 3
bbnam.bf, 8
blockmodel, 11
blockmodel.expand, 13
bonpow, 14
centralgraph, 16
centralization, 17
closeness, 19
component.dist, 20
components, 22
connectedness, 23
cugtest, 26
degree, 28
diag.remove, 30
dyad.census, 31
efficiency, 32
equiv.clust, 33
eval.edgeperturbation, 35
evcent, 36
event2dichot, 38
gclust.centralgraph, 42
gden, 46
gedist, 50
gliop, 51
graphcent, 72
grecip, 73
gvectorize, 80
hierarchy, 83
inforcent, 84
interval.graph, 86
isolates, 90
lab.optimize, 91
lower.tri.remove, 97
lubness, 98
mutuality, 100
netcancor, 101
netlm, 104
netlogit, 105
npostpred, 107
nties, 108
numperm, 109
prestige, 118
qaptest, 135
rgraph, 141
sdmat, 148
sedist, 150
sna.operators, 153
sr2css, 154
stackcount, 155
stresscent, 155
structdist, 157
summary.bayes.factor, 159
summary.bbnam, 160
summary.blockmodel, 161
summary.cugtest, 161
summary.netcancor, 163
summary.netlm, 163
summary.netlogit, 164
summary.qaptest, 165
symmetrize, 165
triad.census, 166
triad.classify, 167
upper.tri.remove, 169

*Topic methods

summary.lnam, 162

*Topic misc

sna-deprecated, 152

*Topic models

bbnam, 3
bbnam.bf, 8
npostpred, 107
potscalered.mcmc, 117
pstar, 132

*Topic multivariate

gcor, 43
gcov, 45
gscor, 74
gscov, 77
hdist, 81
lnam, 94
netcancor, 101
pstar, 132
sdmat, 148
structdist, 157

*Topic optimize

lab.optimize, 91

*Topic print

print.bayes.factor, 120
print.bbnam, 121
print.blockmodel, 122
print.cugtest, 122
print.lnam, 123
print.netcancor, 124
print.netlm, 124
print.netlogit, 125
print.qaptest, 126
print.summary.bayes.factor, 126
print.summary.bbnam, 127
print.summary.blockmodel, 128
print.summary.cugtest, 128
print.summary.lnam, 129
print.summary.netcancor, 130
print.summary.netlm, 130
print.summary.netlogit, 131
print.summary.qaptest, 132

*Topic regression
lnam, 94
netlm, 104
netlogit, 105
pstar, 132

*Topic univar
betweenness, 9
bonpow, 14
centralization, 17
closeness, 19
connectedness, 23
degree, 28
efficiency, 32
evcent, 36
gcor, 43
gcov, 45
gden, 46
gphcent, 72
grecip, 73
gscor, 74
gscov, 77
hdist, 81
hierarchy, 83
infocent, 84
lubness, 98
mutuality, 100
nties, 108
potscaled.mcmc, 117
prestige, 118
sdmat, 148
stresscent, 155
structdist, 157

*Topic utilities
gliop, 51
stackcount, 155

%c (sna.operators), 153
add.isolates, 1, 2, 89, 90
addisolates (sna-deprecated), 152
addisolates-deprecated, 2
apply, 40, 107
arrows, 56, 57
bbnam, 3, 8, 9, 25, 26, 107, 108, 110, 111, 118, 121, 127, 154, 160
bbnam.bf, 7, 8, 121, 160
bbnam.jntlik (sna-internal), 152
bbnam.probtie (sna-internal), 152
betweenness, 9, 156
blockmodel, 11, 14, 34, 112, 122, 151, 161
blockmodel.expand, 12, 13
bonpow, 14, 37, 86
boxplot, 41
cancor, 102, 103
centralgraph, 16, 26, 42, 82
centralization, 9, 11, 14, 16, 17, 19, 20, 28, 29, 36, 37, 72, 73, 84, 86, 118, 120, 155, 157
closeness, 19, 72, 86
cmdscale, 49, 61, 71, 151
coeff.lnam (lnam), 94
component.dist, 20, 22, 23, 51, 88
components, 22, 22, 51, 88
connectedness, 23, 23, 24, 32, 33, 83, 84, 98, 99
consensus, 24
cor, 44
cov, 45
cugtest, 18, 26, 44, 45, 51, 52, 76, 78, 80, 82, 102–104, 106, 113, 122, 123, 134, 137, 149, 158, 162
cutree, 11, 12, 41, 42
degree, 18, 28, 60
Deprecaed, 152
diag, 30
diag.remove, 30, 98, 170
dist, 47, 148
dyad.census, 31, 84, 143, 144, 167
efficiency, 23, 24, 32, 32, 33, 83, 84, 98, 99
eigen, 61, 71
equiv.clust, 12, 33, 41, 113, 114, 151
eval.edgeperturbation, 35, 134
evcent, 16, 36, 86
event2dichot, 7, 38
rgws, 144
rmperm, 110, 142, 146, 147
rperm, 110, 146, 147
sapply, 40
scan, 140
sdmat, 83, 94, 148, 159
se.lnam (lnam), 94
sedist, 33, 34, 41, 150
segments, 56, 57
sna-deprecated, 152
sna-internal, 152
sna.operators, 153
solve, 15, 85
sr2css, 154
stackcount, 155
stresscent, 155
structdist, 82, 83, 93, 94, 109, 149, 157
summary.bayes.factor, 127, 159
summary.bbnam, 160
summary.blockmodel, 128, 161
summary.cugtest, 27, 129, 161
summary.lnam, 129, 162
summary.netcancor, 102, 130, 163
summary.netlm, 105, 131, 163
summary.netlogit, 106, 131, 164
summary.qaptest, 132, 165
symmetrize, 22, 23, 58, 69, 74, 85, 165
triad.census, 31, 166, 167–169
triad.classify, 80, 167, 167
upper.tri, 170
upper.tri.remove, 30, 98, 169