Package ‘sna’

December 10, 2019

Version 2.5

Date 2019-12-09

Title Tools for Social Network Analysis

Depends R (>= 2.0.0), utils, statnet.common, network

Suggests rgl, numDeriv, SparseM

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

License GPL (>= 2)

URL http://www.statnet.org

NeedsCompilation yes

Author Carter T. Butts [aut, cre, cph]

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

Repository CRAN

Date/Publication 2019-12-10 11:00:02 UTC

R topics documented:

add.isolates *Add Isolates to a Graph*

Description

Adds n isolates to the graph (or graphs) in dat.

Usage

```
add.isolates(dat, n, return.as.edgelist = FALSE)
```

Arguments

dat	one or more input graphs.
n	the number of isolates to add.
return.as.edgelist	logical; should the input graph be returned as an edgelist (rather than an adjacency matrix)?

Details

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

Value

The updated graph(s).

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

Butts, C.T., and Carley, K.M. (2001). “Multivariate Methods for Inter-Structural Analysis.” CASOS Working Paper, Carnegie Mellon University.

See Also

isolates

Examples

```
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
```

---

bbnam *Butts' (Hierarchical) Bayesian Network Accuracy Model*

Description

Takes posterior draws from Butts' bayesian network accuracy/estimation model for multiple participant/observers (conditional on observed data and priors), using a Gibbs sampler.

Usage

```
bbnam(dat, model="actor", ...)
bbnam.fixed(dat, nprior=0.5, em=0.25, ep=0.25, diag=FALSE,
    mode="digraph", draws=1500, outmode="draws", anames=NULL,
    onames=NULL)
bbnam.pooled(dat, nprior=0.5, emprior=c(1,11), epprior=c(1,11),
    diag=FALSE, mode="digraph", reps=5, draws=1500, burntime=500,
    quiet=TRUE, anames=NULL, onames=NULL, compute.sqrtrhat=TRUE)
bbnam.actor(dat, nprior=0.5, emprior=c(1,11), epprior=c(1,11),
    diag=FALSE, mode="digraph", reps=5, draws=1500, burntime=500,
    quiet=TRUE, anames=NULL, onames=NULL, compute.sqrtrhat=TRUE)
```

Arguments

`dat`	Input networks to be analyzed. This may be supplied in any reasonable form, but must be reducible to an array of dimension $m \times n \times n$, where n is $\|V(G)\|$, the first dimension indexes the observer (or information source), 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 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.) Non-matrix values will be coerced/expanded to matrix form as appropriate. If no network prior is provided, an uninformative prior on the space of networks will be assumed (i.e., $\Pr(i \to j) = 0.5$). 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 (α, β) pair for the pooled model, and of an $n \times 2$ matrix of (α, β) pairs for the actor model (or something which can be coerced to this form). If no `emprior` is given, a weakly informative prior (1,11) will be assumed; note that this may be 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 (α, β) pair for the pooled model, and of an $n \times 2$ matrix of (α, β) pairs for the actor model (or something which can be coerced to this form). If no `epprior` is given, a weakly informative prior (1,11) will be assumed; note that this may be 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).

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 input data.

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 participants/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 ($\sqrt{\hat{R}}$), 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 1.2 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* $\Pr(j \to k)$). 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). Currently, the default error parameter prior is Beta(1,11), which is both diffuse and which renders negatively informative observers extremely improbable (i.e., on the order of 1e-6). Another 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.) For network priors, a reasonable starting point can often be derived by considering the expected mean degree of the criterion graph: if d represents the user's prior expectation for the mean degree, then $d/(N-1)$ is a natural starting point for the cell values of `nprior`. Butts (2003) discusses a number of issues related to choice of priors for the `bbnam` model, 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

Butts, C. T. (2003). “Network Inference, Error, and Informant (In)Accuracy: A Bayesian Approach.” *Social Networks*, 25(2), 103-140.

Gelman, A.; Carlin, J.B.; Stern, H.S.; and Rubin, D.B. (1995). *Bayesian Data Analysis.* London: Chapman and Hall.

Gelman, A., and Rubin, D.B. (1992). “Inference from Iterative Simulation Using Multiple Sequences.” *Statistical Science,* 7, 457-511.

Krackhardt, D. (1987). “Cognitive Social Structures.” *Social Networks,* 9, 109-134.

See Also

npostpred, event2dichot, bbnam.bf

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(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=0.5, em.fp=0.5, ep.fp=0.5, emprior.pooled=c(1, 11),
    epprior.pooled=c(1, 11), emprior.actor=c(1, 11), epprior.actor=c(1, 11),
    diag=FALSE, mode="digraph", reps=1000)
```

Arguments

- `dat` Input networks to be analyzed. This may be supplied in any reasonable form, but must be reducible to an array of dimension $m \times n \times n$, where n is $|V(G)|$, the first dimension indexes the observer (or information source), 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 draws 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.) Non-matrix values will be coerced/expanded to matrix form as appropriate. If no network prior is provided, an uninformative prior on the space of networks will be assumed (i.e., $\Pr(i \to j) = 0.5$). 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
- `emprior.pooled` (α, β) pairs for the (beta) false negative prior under the pooled model
- `epprior.pooled` (α, β) pairs for the (beta) false positive prior under the pooled model
- `emprior.actor` Matrix of per observer (α, β) pairs for the (beta) false negative prior under the per observer/actor model, or something that can be coerced to this form
- `epprior.actor` Matrix of per observer $((\alpha, \beta)$ pairs for the (beta) false positive prior under the per observer/actor model, or something that can be coerced to this form
- `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 log 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.

By default, bbnam.bf uses weakly informative Beta(1,11) priors for false positive and false negative rates, which may not be appropriate for all cases. (Likewise, the initial network prior is uniformative.) Users are advised to consider adjusting the error rate priors when using this function in a practical context; for instance, it is often reasonable to expect higher false negative rates (on average) than false positive rates, and to expect the criterion graph density to be substantially less than 0.5. See the reference below for a discussion of this issue.

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

Butts, C. T. (2003). "Network Inference, Error, and Informant (In)Accuracy: A Bayesian Approach." *Social Networks*, 25(2), 103-140.

Robert, C. (1994). *The Bayesian Choice: A Decision-Theoretic Motivation.* Springer.

See Also

bbnam

Examples

```
#Create some random data from the "pooled" model
g<-rgraph(7)
g.p<-0.8*g+0.2*(1-g)
dat<-rgraph(7,7,tprob=g.p)

#Estimate the log Bayes Factors
b<-bbnam.bf(dat,emprior.pooled=c(3,5),epprior.pooled=c(3,5),
    emprior.actor=c(3,5),epprior.actor=c(3,5))
#Print the results
b
```

betweenness *Compute the Betweenness Centrality Scores of Network Positions*

Description

betweenness takes one or more graphs (dat) and returns the betweenness centralities of positions (selected by nodes) within the graphs indicated by g. 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=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, cmode="directed", geodist.precomp=NULL,
    rescale=FALSE, ignore.eval=TRUE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.
nodes	vector 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, or a variant form – see below).
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.
ignore.eval	logical; ignore edge values when computing shortest paths?

Details

The shortest-path betweenness of a vertex, v, is given by

$$C_B(v) = \sum_{i,j:i\neq j,i\neq v,j\neq v} \frac{g_{ivj}}{g_{ij}}$$

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.”

Several variant forms of shortest-path betweenness exist, and can be selected using the `cmode` argument. Supported options are as follows:

`directed` Standard betweenness (see above), calculated on directed pairs. (This is the default option.)

`undirected` Standard betweenness (as above), calculated on undirected pairs (undirected graphs only).

`endpoints` Standard betweenness, with direct connections counted towards ego’s score. This expresses the intuition that individuals’ control over their own direct contacts should be considered in their total score (e.g., when betweenness is interpreted as a measure of information control).

`proximalsrc` Borgatti’s *proximal source betweenness,* given by

$$C_B(v) = \sum_{i,j: i \neq v, i \neq j, j \to v} \frac{g_{ivj}}{g_{ij}}.$$

This variant allows betweenness to accumulate only for the last intermediating vertex in each incoming geodesic; this expresses the notion that, by serving as the “proximal source” for the target, this particular intermediary will in some settings have greater influence or control than other intervening parties.

`proximaltar` Borgatti’s *proximal target betweenness,* given by

$$C_B(v) = \sum_{i,j: i \neq v, i \to v, i \neq j} \frac{g_{ivj}}{g_{ij}}.$$

This counterpart to proximal source betweenness (above) allows betweenness to accumulate only for the first intermediating vertex in each outgoing geodesic; this expresses the notion that, by serving as the “proximal target” for the source, this particular intermediary will in some settings have greater influence or control than other intervening parties.

`proximalsum` The sum of Borgatti’s proximal source and proximal target betweenness scores (above); this may be used when either role is regarded as relevant to the betweenness calculation.

`lengthscaled` Borgetti and Everett’s *length-scaled betweenness*, given by

$$C_B(v) = \sum_{i,j: i \neq j, i \neq v, j \neq v} \frac{1}{d_{ij}} \frac{g_{ivj}}{g_{ij}},$$

where d_{ij} is the geodesic distance from i to j. This measure adjusts the standard betweenness score by downweighting long paths (e.g., as appropriate in circumstances for which such paths are less-often used).

`linearscaled` Geisberger et al.’s *linearly-scaled betweenness*:

$$C_B(v) = \sum_{i,j: i \neq j, i \neq v, j \neq v} \frac{1}{d_{ij}} \frac{g_{ivj}}{g_{ij}}.$$

This variant modifies the standard betweenness score by giving more weight to intermediaries which are closer to their targets (much like proximal source betweenness, above). This may be of use when those near the end of a path have greater direct control over the flow of influence or resources than those near its source.

See Brandes (2008) for details and additional references. Geodesics for all of the above can be calculated using valued edges by setting `ignore.eval=TRUE`. Edge values are interpreted as distances for this purpose; proximity data should be transformed accordingly before invoking this routine.

Value

A vector, matrix, or list containing the betweenness scores (depending on the number and size of the input graphs).

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

Borgatti, S.P. and Everett, M.G. (2006). "A Graph-Theoretic Perspective on Centrality." *Social Networks*, 28, 466-484.

Brandes, U. (2008). "On Variants of Shortest-Path Betweenness Centrality and their Generic Computation." *Social Networks*, 30, 136–145.

Freeman, L.C. (1979). "Centrality in Social Networks I: Conceptual Clarification." *Social Networks*, 1, 215-239.

Geisberger, R., Sanders, P., and Schultes, D. (2008). "Better Approximation of Betweenness Centrality." In *Proceedings of the 10th Workshop on Algorithm Engineering and Experimentation (ALENEX'08)*, 90-100. SIAM.

See Also

centralization, stresscent, geodist

Examples

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

bicomponent.dist *Calculate the Bicomponents of a Graph*

Description

bicomponent.dist returns the bicomponents of an input graph, along with size distribution and membership information.

Usage

```
bicomponent.dist(dat, symmetrize = c("strong", "weak"))
```

Arguments

dat	a graph or graph stack.
symmetrize	symmetrization rule to apply when pre-processing the input (see symmetrize).

Details

The bicomponents of undirected graph G are its maximal 2-connected vertex sets. bicomponent.dist calculates the bicomponents of G, after first coercing to undirected form using the symmetrization rule in symmetrize. In addition to bicomponent memberships, various summary statistics regarding the bicomponent distribution are returned; see below.

Value

A list containing

members	A list, with one entry per bicomponent, containing component members.
memberships	A vector of component memberships, by vertex. (Note: memberships may not be unique.) Vertices not belonging to any bicomponent have membership values of NA.
csize	A vector of component sizes, by bicomponent.
cdist	A vector of length $\|V(G)\|$ with the (unnormalized) empirical distribution function of bicomponent sizes.

Note

Remember that bicomponents can intersect; when this occurs, the relevant vertices' entries in the membership vector are assigned to one of the overlapping bicomponents on an arbitrary basis. The members element of the return list is the safe way to recover membership information.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Brandes, U. and Erlebach, T. (2005). *Network Analysis: Methodological Foundations.* Berlin: Springer.

See Also

`component.dist`, `cutpoints`, `symmetrize`

Examples

```
#Draw a moderately sparse graph
g<-rgraph(25,tp=2/24,mode="graph")

#Compute the bicomponents
bicomponent.dist(g)
```

---

blockmodel — *Generate Blockmodels Based on Partitions of Network Positions*

Description

Given a set of equivalence classes (in the form of an `equiv.clust` object, `hclust` object, or membership vector) and one or more graphs, `blockmodel` will form a blockmodel of the input graph(s) based on the classes in question, using the specified block content type.

Usage

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

Arguments

`dat`	one or more input graphs.
`ec`	equivalence classes, in the form of an object of class `equiv.clust` or `hclust`, or a membership vector.
`k`	the number of classes to form (using `cutree`).
`h`	the height at which to split classes (using `cutree`).
`block.content`	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

Unless a vector of classes is specified, 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.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Doreian, P.; Batagelj, V.; and Ferligoj, A. (2005). *Generalized Blockmodeling.* Cambridge: Cambridge University Press.

White, H.C.; Boorman, S.A.; and Breiger, R.L. (1976). "Social Structure from Multiple Networks I: Blockmodels of Roles and Positions." *American Journal of Sociology*, 81, 730-779.

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.

Details

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 interclass 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

Doreian, P.; Batagelj, V.; and Ferligoj, A. (2005). *Generalized Blockmodeling.* Cambridge: Cambridge University Press.

White, H.C.; Boorman, S.A.; and Breiger, R.L. (1976). "Social Structure from Multiple Networks I: Blockmodels of Roles and Positions." *American Journal of Sociology*, 81, 730-779.

See Also

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)

#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
```

---

bn *Fit a Biased Net Model*

Description

Fits a biased net model to an input graph, using moment-based or maximum pseudolikelihood techniques.

Usage

```
bn(dat, method = c("mple.triad", "mple.dyad", "mple.edge",
    "mtle"), param.seed = NULL, param.fixed = NULL,
    optim.method = "BFGS", optim.control = list(),
    epsilon = 1e-05)
```

Arguments

`dat`	a single input graph.
`method`	the fit method to use (see below).
`param.seed`	seed values for the parameter estimates.
`param.fixed`	parameter values to fix, if any.
`optim.method`	method to be used by `optim`.
`optim.control`	control parameter for `optim`.
`epsilon`	tolerance for convergence to extreme parameter values (i.e., 0 or 1).

Details

The biased net model stems from early work by Rapoport, who attempted to model networks via a hypothetical "tracing" process. This process may be described loosely as follows. One begins with a small "seed" set of vertices, each member of which is assumed to nominate (generate ties to) other members of the population with some fixed probability. These members, in turn, may nominate new members of the population, as well as members who have already been reached. Such nominations may be "biased" in one fashion or another, leading to a non-uniform growth process. Specifically, let e_{ij} be the random event that vertex i nominates vertex j when reached. Then the conditional probability of e_{ij} is given by

$$\Pr(e_{ij}|T) = 1 - (1 - \Pr(B_e)) \prod_k (1 - \Pr(B_k|T))$$

where T is the current state of the trace, B_e is the a Bernoulli event corresponding to the baseline probability of e_{ij}, and the B_k are "bias events." Bias events are taken to be independent Bernoulli trials, given T, such that e_{ij} is observed with certainty if any bias event occurs. The specification of a biased net model, then, involves defining the various bias events (which, in turn, influence the structure of the network).

Although other events have been proposed, the primary bias events employed in current biased net models are the "parent bias" (a tendency to return nominations); the "sibling bias" (a tendency to nominate alters who were nominated by the same third party); and the "double role bias" (a tendency to nominate alters who are both siblings and parents). These bias events, together with the baseline edge events, are used to form the standard biased net model. It is standard to assume homogeneity within bias class, leading to the four parameters π (probability of a parent bias event), σ (probability of a sibling bias event), ρ (probability of a double role bias event), and d (probability of a baseline event).

Unfortunately, there is no simple expression for the likelihood of a graph given these parameters (and hence, no basis for likelihood based inference). However, Skvoretz et al. have derived a class of maximum pseudo-likelihood estimators for the the biased net model, based on local approximations to the likelihood at the edge, dyad, or triad level. These estimators may be employed within `bn` by selecting the appropriate MPLE for the *method* argument. Alternately, it is also possible to derive expected triad census rates for the biased net model, allowing an estimator which maximizes the likelihood of the observed triad census (essentially, a method of moments procedure). This last may be selected via the argument `mode="mtle"`. In addition to estimating model parameters, `bn` generates predicted edge, dyad, and triad census statistics, as well as structure statistics (using the Fararo-Sunshine recurrence). These can be used to evaluate goodness-of-fit.

print, summary, and plot methods are available for bn objects. See rgbn for simulation from biased net models.

Value

An object of class bn.

Note

Asymptotic properties of the MPLE are not known for this model. Caution is strongly advised.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Fararo, T.J. and Sunshine, M.H. (1964). “A study of a biased friendship net.” Syracuse, NY: Youth Development Center.

Rapoport, A. (1957). “A contribution to the theory of random and biased nets.” *Bulletin of Mathematical Biophysics,* 15, 523-533.

Skvoretz, J.; Fararo, T.J.; and Agneessens, F. (2004). “Advances in biased net theory: definitions, derivations, and estimations.” *Social Networks,* 26, 113-139.

See Also

rgbn, structure.statistics

Examples

```
#Generate a random graph
g<-rgraph(25)

#Fit a biased net model, using the triadic MPLE
gbn<-bn(g)

#Examine the results
summary(gbn)
plot(gbn)

#Now, fit a model containing only a density parameter
gbn<-bn(g,param.fixed=list(pi=0,sigma=0,rho=0))
summary(gbn)
plot(gbn)
```

bonpow *Find Bonacich Power Centrality Scores of Network Positions*

Description

bonpow takes one or more graphs (dat) and returns the Boncich power centralities of positions (selected by nodes) within the graphs indicated by g. 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=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, exponent=1, rescale=FALSE, tol=1e-07)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.
nodes	vector 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.
exponent	exponent (decay rate) for the Bonacich power centrality score; can be negative
rescale	if true, centrality scores 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 $C_{BP}(\alpha,\beta) = \alpha\left(\mathbf{I}-\beta\mathbf{A}\right)^{-1}\mathbf{A}\mathbf{1}$, where β is an attenuation parameter (set here by exponent) and $\mathbf{A}$ is the graph adjacency matrix. (The coefficient α 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 $\beta \rightarrow 1/\lambda_{\mathbf{A}1}$ (the reciprocal of the largest eigenvalue of $\mathbf{A}$), this is to within a constant multiple of the familiar eigenvector centrality score; for other values of β, the behavior of the measure is quite different. In particular, β gives positive and negative weight to even and odd walks, respectively, as can be seen from the series expansion $C_{BP}(\alpha,\beta) = \alpha\sum_{k=0}^{\infty}\beta^k\mathbf{A}^{k+1}\mathbf{1}$ which converges so long as $|\beta| < 1/\lambda_{\mathbf{A}1}$. The magnitude of β

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, matrix, or list containing the centrality scores (depending on the number and size of the input graphs).

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 <buttsc@uci.edu>

References

Bonacich, P. (1972). "Factoring and Weighting Approaches to Status Scores and Clique Identification." *Journal of Mathematical Sociology*, 2, 113-120.

Bonacich, P. (1987). "Power and Centrality: A Family of Measures." *American Journal of Sociology*, 92, 1170-1182.

See Also

`centralization`, `evcent`

Examples

```
#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)
```

---

brokerage *Perform a Gould-Fernandez Brokerage Analysis*

Description

Performs the brokerage analysis of Gould and Fernandez on one or more input graphs, given a class membership vector.

Usage

```
brokerage(g, cl)
```

Arguments

g	one or more input graphs.
cl	a vector of class memberships.

Details

Gould and Fernandez (following Marsden and others) describe *brokerage* as the role played by a social actor who mediates contact between two alters. More formally, vertex v is a broker for distinct vertices a and b iff $a \to v \to b$ and $a \not\to b$. Where actors belong to a priori distinct groups, group membership may be used to segment brokerage roles into particular types. Let $A \to B \to C$ denote the two-path associated with a brokerage structure, such that some vertex from group B brokers the connection from some vertex from group A to a vertex in group C. The types of brokerage roles defined by Gould and Fernandez (and their accompanying two-path structures) are then defined in terms of group membership as follows:

- w_I: Coordinator role; the broker mediates contact between two individuals from his or her own group. Two-path structure: $A \to A \to A$
- w_O: Itinerant broker role; the broker mediates contact between two individuals from a single group to which he or she does not belong. Two-path structure: $A \to B \to A$
- b_{OI}: Gatekeeper role; the broker mediates an incoming contact from an out-group member to an in-group member. Two-path structure: $A \to B \to B$
- b_{IO}: Representative role; the broker mediates an outgoing contact from an in-group member to an out-group member. Two-path structure: $A \to A \to B$
- b_O: Liaison role; the broker mediates contact between two individuals from different groups, neither of which is the group to which he or she belongs. Two-path structure: $A \to B \to C$
- t: Total (cumulative) brokerage role occupancy. (Any of the above two-paths.)

The *brokerage score* for a given vertex with respect to a given role is the number of ordered pairs having the appropriate group membership(s) brokered by said vertex. `brokerage` computes the brokerage scores for each vertex, given an input graph and vector of class memberships. Aggregate scores are also computed at the graph level, which correspond to the total frequency of each role type within the network structure. Expectations and variances of the brokerage scores conditional on size and density are computed, along with approximate z-tests for incidence of brokerage. (Note that the accuracy of the normality assumption is not known in the general case; see Gould and Fernandez (1989) for details. Simulation-based tests may be desirable as an alternative.)

Value

An object of class `brokerage`, containing the following elements:

`raw.nli`	The matrix of observed brokerage scores, by vertex
`exp.nli`	The matrix of expected brokerage scores, by vertex
`sd.nli`	The matrix of predicted brokerage score standard deviations, by vertex
`z.nli`	The matrix of standardized brokerage scores, by vertex
`raw.gli`	The vector of observed aggregate brokerage scores
`exp.gli`	The vector of expected aggregate brokerage scores
`sd.gli`	The vector of predicted aggregate brokerage score standard deviations
`z.gli`	The vector of standardized aggregate brokerage scores
`exp.grp`	The matrix of expected brokerage scores, by group
`sd.grp`	The matrix of predicted brokerage score standard deviations, by group
`cl`	The vector of class memberships
`clid`	The original class names
`n`	The input class sizes
`N`	The order of the input network

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Gould, R.V. and Fernandez, R.M. 1989. “Structures of Mediation: A Formal Approach to Brokerage in Transaction Networks.” *Sociological Methodology,* 19: 89-126.

See Also

triad.census, gtrans

Examples

```
#Draw a random network with 3 groups
g<-rgraph(15)
cl<-rep(1:3,5)

#Compute a brokerage object
b<-brokerage(g,cl)
summary(b)
```

---

centralgraph *Find the Central Graph of a Labeled Graph Stack*

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

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

Arguments

`dat`	one or more input graphs.
`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

Banks, D.L., and Carley, K.M. (1994). “Metric Inference for Social Networks.” *Journal of Classification*, 11(1), 121-49.

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=NULL, mode="digraph", diag=FALSE,
    normalize=TRUE, ...)
```

Arguments

`dat`	one or more input graphs.
`FUN`	Function to return nodal centrality scores.
`g`	Integer indicating the index of the graph for which centralization should be computed. By default, all graphs are employed.
`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

Freeman, L.C. (1979). "Centrality in Social Networks I: Conceptual Clarification." *Social Networks*, 1, 215-239.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

cugtest

Examples

```
#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)
```

clique.census *Compute Cycle Census Information*

Description

clique.census computes clique census statistics on one or more input graphs. In addition to aggregate counts of maximal cliques, results may be disaggregated by vertex and co-membership information may be computed.

Usage

```
clique.census(dat, mode = "digraph", tabulate.by.vertex = TRUE,
    clique.comembership = c("none", "sum", "bysize"), enumerate = TRUE,
    na.omit = TRUE)
```

Arguments

dat	one or more input graphs.
mode	"digraph" for directed graphs, or "graph" for undirected graphs.
tabulate.by.vertex	logical; should maximal clique counts be tabulated by vertex?
clique.comembership	the type of clique co-membership information to be tabulated, if any. "sum" returns a vertex by vertex matrix of clique co-membership counts; these are disaggregated by clique size if "bysize" is used. If "none" is given, no co-membership information is computed.
enumerate	logical; should an enumeration of all maximal cliques be returned?
na.omit	logical; should missing edges be omitted?

Details

A (maximal) *clique* is a maximal set of mutually adjacenct vertices. Cliques are important for their role as cohesive subgroups, but show up in many other contexts as well.

A *subgraph census statistic* is a function which, for any given graph and subgraph, gives the number of copies of the latter contained in the former. A collection of subgraph census statistics is referred to as a *subgraph census*; widely used examples include the dyad and triad censuses, implemented in sna by the dyad.census and triad.census functions (respectively). Likewise, kpath.census and kcycle.census compute a range of census statistics related to k-paths and k-cycles. clique.census provides similar functionality for the census of maximal cliques, including:

- Aggregate counts of maximal cliques by size.
- Counts of cliques to which each vertex belongs (when tabulate.byvertex==TRUE).
- Counts of clique co-memberships, potentially disaggregated by size (when the appropriate co-membership argument is set to bylength).

These calculations are intrinsically expensive (clique enumeration is NP hard in the general case), and users should be aware that computing the census can be impractical on large graphs (unless they are very sparse). On the other hand, the algorithm employed here (a variant of Makino and Uno (2004)) is generally fast enough to suport enumeration for even dense graphs of several hundred vertices on a typical desktop computer.

Calling this function with `mode=="digraph"`, forces and initial symmetrization step, which can be avoided with `mode=="graph"`. While incorrectly employing the default is harmless (except for the relatively small cost of verifying symmetry), setting `mode=="graph"` incorrectly may result in problematic behavior. When in doubt, stick with the default.

Value

A list with the following elements:

`clique.count` If `tabulate.byvertex==FALSE`, a vector of aggregate counts by clique size. Otherwise, a matrix whose first column is a vector of aggregate clique counts, and whose succeeding columns contain vectors of clique counts for each vertex.

`clique.comemb` If `clique.comembership!="none"`, a matrix or array containing co-membership in cliques by vertex pairs. If `clique.comembership=="sum"`, only a matrix of co-memberships is returned; if `bysize` is used, however, co-memberships are returned in a `maxsize` by n by n array whose i, j, kth cell is the number of cliques of size i containing `j` and `k` (with `maxsize` being the size of the largest maximal clique).

`cliques` If `enumerate=TRUE`, a list of length equal to the maximum clique size, each element of which is in turn a list of all cliques of corresponding size (given as vectors of vertices).

Warning

The computational cost of calculating cliques grows very sharply in size and network density. It is possible that the expected completion time for your calculation may exceed your life expectancy (and those of subsequent generations).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

Makino, K. and Uno, T. (2004.) “New Algorithms for Enumerating All Maximal Cliques.” In T. Hagerup and J. Katajainen (eds.), *SWAT 2004*, LNCS 3111, 260-272. Berlin: Springer-Verlag.

See Also

`dyad.census`, `triad.census`, `kcycle.census`, `kpath.census`

Examples

```
#Generate a fairly dense random graph
g<-rgraph(25)

#Obtain cliques by vertex, with co-membership by size
cc<-clique.census(g,clique.comembership="bysize")
cc$clique.count                              #Examine clique counts
cc$clique.comemb[1,,]                        #Isolate co-membership is trivial
cc$clique.comemb[2,,]                        #Co-membership for 2-cliques
cc$clique.comemb[3,,]                        #Co-membership for 3-cliques
cc$cliques                                   #Enumerate the cliques
```

---

closeness *Compute the Closeness Centrality Scores of Network Positions*

Description

closeness takes one or more graphs (dat) and returns the closeness centralities of positions (selected by nodes) within the graphs indicated by g. 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=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, cmode="directed", geodist.precomp=NULL,
    rescale=FALSE, ignore.eval=TRUE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). 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 pairs, or an alternate measure).

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.

ignore.eval
: logical; should edge values be ignored when calculating geodesics?

Details

The closeness of a vertex v is defined as

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

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 `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.

An alternate form of closeness (apparently due to Gil and Schmidt (1996)) is obtained by taking the sum of the inverse distances to each vertex, i.e.

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

This measure correlates well with the standard form of closeness where both are well-defined, but lacks the latter's pathological behavior on disconnected graphs. Computation of alternate closeness may be performed via the argument `cmode="suminvdir"` (directed case) and `cmode="suminvundir"` (undirected case). The corresponding arguments `cmode="directed"` and `cmode="undirected"` return the standard closeness scores in the directed or undirected cases (respectively). Although treated here as a measure of closeness, this index was originally intended to capture power or efficacy; in its original form, the Gil-Schmidt power index is a renormalized version of the above. Specifically, let $R(v,G)$ be the set of vertices reachable by v in $V \setminus v$. Then the Gil-Schmidt power index is defined as

$$C_{GS}(v) = \frac{\sum_{i \in R(v,G)} \frac{1}{d(v,i)}}{|R(v,G)|}.$$

with C_{GS} defined to be 0 for vertices with no outneighbors. This may be obtained via the argument `cmode="gil-schmidt"`.

Value

A vector, matrix, or list containing the closeness scores (depending on the number and size of the input graphs).

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

Freeman, L.C. (1979). "Centrality in Social Networks I: Conceptual Clarification." *Social Networks*, 1, 215-239.

Gil, J. and Schmidt, S. (1996). "The Origin of the Mexican Network of Power". Proceedings of the International Social Network Conference, Charleston, SC, 22-25.

Sinclair, P.A. (2009). "Network Centralization with the Gil Schmidt Power Centrality Index" *Social Networks*, 29, 81-92.

See Also

centralization

Examples

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

---

coleman — *Coleman's High School Friendship Data*

Description

James Coleman (1964) reports research on self-reported friendship ties among 73 boys in a small high school in Illinois over the 1957-1958 academic year. Networks of reported ties for all 73 informants are provided for two time points (fall and spring).

Usage

```
data(coleman)
```

Format

An adjacency array containing two directed, unvalued 73-node networks:

[1,,]	Fall	binary matrix	Friendship for Fall, 1957
[2,,]	Spring	binary matrix	Friendship for Spring, 1958

Details

Both networks reflect answers to the question, "What fellows here in school do you go around with most often?" with the presence of an (i, j, k) edge indicating that j nominated k in time period i. The data are unvalued and directed; although the self-reported ties are highly reciprocal, unreciprocated nominations are possible.

It should be noted that, although this data is usually described as "friendship," the sociometric item employed might be more accurately characterized as eliciting "frequent elective interaction." This should be borne in mind when interpreting this data.

References

Coleman, J. S. (1964). *Introduction to Mathermatical Sociology.* New York: Free Press.

Examples

```
data(coleman)

#Plot showing edges by time point
gplot(coleman[1,,]|coleman[2,,],edge.col=2*coleman[1,,]+3*coleman[2,,])
```

`component.dist` *Calculate the Component Size Distribution of a Graph*

Description

`component.dist` returns a list containing a vector of length `n` such that the `i`th element contains the number of components of graph G having size `i`, and a vector of length `n` giving component membership (where `n` is the graph order). Component strength is determined by the `connected` parameter; see below for details.

`component.largest` identifies the component(s) of maximum order within graph `G`. It returns either a `logical` vector indicating membership in a maximum component or the adjacency matrix of the subgraph of G induced by the maximum component(s), as determined by `result`. Component strength is determined as per `component.dist`.

Usage

```
component.dist(dat, connected=c("strong","weak","unilateral",
     "recursive"))

component.largest(dat, connected=c("strong","weak","unilateral",
     "recursive"), result = c("membership", "graph"), return.as.edgelist =
     FALSE)
```

Arguments

`dat`	one or more input graphs.
`connected`	a string selecting strong, weak, unilateral or recursively connected components; by default, `"strong"` components are used.
`result`	a string indicating whether a vector of membership indicators or the induced subgraph of the component should be returned.

return.as.edgelist
: logical; if `result=="graph"`, should the resulting structure be returned in edgelist form?

Details

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 `dat` is symmetric, then the `connected` parameter has no effect.

Value

For `component.dist`, a list 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

If multiple input graphs are given, the return value is a list of lists.

For `component.largest`, either a `logical` vector of component membership indicators or the adjacency matrix/edgelist of the subgraph induced by the largest component(s) is returned. If multiple graphs were given as input, a list of results is returned.

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 that 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. (`component.dist` generates a warning when this occurs.) It is recommended that the `unilateral` option be avoided where possible.

Do not make the mistake of assuming that the subgraphs returned by `component.largest` are necessarily connected. This is *usually* the case, but depends upon the uniqueness of the largest component.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

West, D.B. (1996). *Introduction to Graph Theory.* Upper Saddle River, N.J.: Prentice Hall.

See Also

components, symmetrize, reachability geodist

Examples

```
g<-rgraph(20,tprob=0.06)   #Generate a sparse random graph

#Find weak components
cd<-component.dist(g,connected="weak")
cd$membership              #Who's in what component?
cd$csize                   #What are the component sizes?
                           #Plot the size distribution
plot(1:length(cd$cdist),cd$cdist/sum(cd$cdist),ylim=c(0,1),type="h")
lgc<-component.largest(g,connected="weak")  #Get largest component
gplot(g,vertex.col=2+lgc)  #Plot g, with component membership
                           #Plot largest component itself
gplot(component.largest(g,connected="weak",result="graph"))

#Find strong components
cd<-component.dist(g,connected="strong")
cd$membership              #Who's in what component?
cd$csize                   #What are the component sizes?
                           #Plot the size distribution
plot(1:length(cd$cdist),cd$cdist/sum(cd$cdist),ylim=c(0,1),type="h")
lgc<-component.largest(g,connected="strong")  #Get largest component
gplot(g,vertex.col=2+lgc)  #Plot g, with component membership
                           #Plot largest component itself
gplot(component.largest(g,connected="strong",result="graph"))
```

---

component.size.byvertex

Get Component Sizes, by Vertex

Description

This function computes the component structure of the input network, and returns a vector whose ith entry is the size of the component to which i belongs. This is useful e.g. for studies of diffusion or similar applications.

Usage

```
component.size.byvertex(dat, connected = c("strong", "weak",
    "unilateral", "recursive"))
```

Arguments

`dat`	one or more input graphs (for best performance, sna edgelists or network objects are suggested).
`connected`	a string selecting the connectedness definition to use; by default, `"strong"` components are used.

Details

Component sizes are here computed using `component.dist`; see this function for additional information.

In an undirected graph, the size of v's component represents the maximum number of nodes that can be reached by a diffusion process along the edges of the graph originating with node v; the expectation of component sizes by vertex (rather than the mean component size) is thus one measure of the maximum average diffusion potential of a graph. Because this quantity is monotone with respect to edge addition, it can be bounded using Bernoulli graphs (see Butts (2011)). In the directed case, multiple types of components are possible; see `component.dist` for details.

Value

A vector of length equal to the number of vertices in `dat`, whose ith element is the number of vertices in the component to which the ith vertex belongs.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

West, D.B. (1996). *Introduction to Graph Theory.* Upper Saddle River, N.J.: Prentice Hall.

Butts, C.T. (2011). "Bernoulli Bounds for General Random Graphs." *Sociological Methodology*, 41, 299-345.

See Also

`component.dist`

Examples

```
#Generate a random undirected graph
g<-rgraph(100,tprob=1.5/99,mode="graph",return.as.edgelist=TRUE)

#Get the component sizes for each vertex
cs<-component.size.byvertex(g)
cs
```

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

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

Arguments

`dat`	one or more input graphs.
`connected`	the the component definition to be used by `component.dist` during component extraction.
`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 other component types can be returned if so desired. (See `component.dist` for details.) For symmetric matrices, this is obviously a moot point.

Value

A vector containing the number of components for each graph in `dat`

Author(s)

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

References

West, D.B. (1996). *Introduction to Graph Theory*. Upper Saddle River, NJ: Prentice Hall.

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 one or more graphs (dat) and returns the Krackhardt connectedness scores for the graphs selected by g.

Usage

```
connectedness(dat, g=NULL)
```

Arguments

dat	one or more graphs.
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 <buttsc@uci.edu>

References

Krackhardt, David. (1994). "Graph Theoretical Dimensions of Informal Organizations." In K. M. Carley and M. J. Prietula (Eds.), *Computational Organization Theory*, 89-111. Hillsdale, NJ: Lawrence Erlbaum and Associates.

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=1e-06, maxiter=1e3, verbose=TRUE, no.bias=FALSE)
```

Arguments

dat	a set of input graphs (must have same order).
mode	"digraph" for directed data, else "graph".
diag	logical; should diagonals (loops) be treated as data?
method	one of "central.graph", "single.reweight", "iterative.reweight", "romney.batchelder", "PCA.reweight", "LAS.intersection", "LAS.union", "OR.row", or "OR.col".
tol	convergence tolerance for the iterative reweighting and B-R algorithms.

maxiter	maximum number of iterations to take (regardless of convergence) for the iterative reweighting and B-R algorithms.
verbose	logical; should bias and competency parameters be reported (where computed)?
no.bias	logical; should responses be assumed to be unbiased?

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. `iterative.reweight`: Similar to `single.reweight`, but the consensus structure and accuracy parameters are estimated via an iterated proportional fitting scheme. The implementation employed here uses both bias and competency parameters.
4. `romney.batchelder`: Fits a Romney-Batchelder informant accuracy model using IPF. This is very similar to `iterative.reweight`, but can be interpreted as the result of a process in which each informant report is correct with a probability equal to the informant’s competency score, and otherwise equal to a Bernoulli trial with parameter equal to the informant’s bias score.
5. `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.
6. `LAS.intersection`: Estimate the consensus structure using the locally aggregated structure (intersection rule). In this model, an i->j edge exists iff i *and* j agree that it exists.
7. `LAS.union`: Estimate the consensus structure using the locally aggregated structure (union rule). In this model, an i->j edge exists iff i *or* j agree that it exists.
8. `OR.row`: Estimate the consensus structure using own report. Here, we take each informant’s outgoing tie reports to be correct.
9. `OR.col`: Estimate the consensus structure using own report. Here, we take each informant’s incoming tie reports to be correct.

Note that the results returned by the single weighting algorithms are not dichotomized by default; since some algorithms thus 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 (or the R-B model) may prove more useful than the ad-hoc solutions.

Value

An adjacency matrix representing the consensus structure

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Banks, D.L., and Carley, K.M. (1994). “Metric Inference for Social Networks.” *Journal of Classification,* 11(1), 121-49.

Butts, C.T., and Carley, K.M. (2001). “Multivariate Methods for Inter-Structural Analysis.” CASOS Working Paper, Carnegie Mellon University.

Krackhardt, D. (1987). “Cognitive Social Structures.” *Social Networks,* 9, 109-134.

Romney, A.K.; Weller, S.C.; and Batchelder, W.H. (1986). “Culture as Consensus: A Theory of Culture and Informant Accuracy.” *American Anthropologist,* 88(2), 313-38.

See Also

bbnam, centralgraph

Examples

```
#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
```

---

cug.test *Univariate Conditional Uniform Graph Tests*

Description

cug.test takes an input network and conducts a conditional uniform graph (CUG) test of the statistic in FUN, using the conditioning statistics in cmode. The resulting test object has custom print and plot methods.

Usage

```
cug.test(dat, FUN, mode = c("digraph", "graph"), cmode = c("size",
    "edges", "dyad.census"), diag = FALSE, reps = 1000,
    ignore.eval = TRUE, FUN.args = list())
```

Arguments

- `dat` one or more input graphs.
- `FUN` the function generating the test statistic; note that this must take a graph as its first argument, and return a single numerical value.
- `mode` `graph` if `dat` is an undirected graph, else `digraph`.
- `cmode` string indicating the type of conditioning to be applied.
- `diag` logical; should self-ties be treated as valid data?
- `reps` number of Monte Carlo replications to use.
- `ignore.eval` logical; should edge values be ignored? (Note: `TRUE` is usually more efficient.)
- `FUN.args` a list containing any additional arguments to `FUN`.

Details

`cug.test` is an improved version of `cugtest`, for use only with univariate CUG hypotheses. Depending on `cmode`, conditioning on the realized size, edge count (or exact edge value distribution), or dyad census (or dyad value distribution) can be selected. Edges are treated as unvalued unless `ignore.eval=FALSE`; since the latter setting is less efficient for sparse graphs, it should be used only when necessary.

A brief summary of the theory and goals of conditional uniform graph testing can be found in the reference below. See also `cugtest` for a somewhat informal description.

Value

An object of class `cug.test`.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, Carter T. (2008). “Social Networks: A Methodological Introduction.” *Asian Journal of Social Psychology,* 11(1), 13–41.

See Also

`cugtest`

Examples

```
#Draw a highly reciprocal network
g<-rguman(1,15,mut=0.25,asym=0.05,null=0.7)

#Test transitivity against size, density, and the dyad census
cug.test(g,gtrans,cmode="size")
cug.test(g,gtrans,cmode="edges")
cug.test(g,gtrans,cmode="dyad.census")
```

cugtest *Perform Conditional Uniform Graph (CUG) Hypothesis Tests for Graph-Level Indices*

Description

cugtest tests an arbitrary GLI (computed on dat by FUN) against a conditional uniform graph null hypothesis, via Monte Carlo simulation. Some variation in the nature of the conditioning is available; currently, conditioning only on size, conditioning jointly on size and estimated tie probability (via expected 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

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

Arguments

dat	graph(s) to be analyzed.
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 $\lvert V(G)\rvert$). Ifcmode=="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 equivalent 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. (Note that maximum entropy distributions conditional on expected statistics are not in general correctly referred to as "conditional uniform graphs", but have been described as such for independent-dyad models; this is indeed the case for this function, although such terminology is not really proper. See `cug.test` for CUG tests with exact conditioning.) 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 observed statistic 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 quantiles.

The `cugtest` procedure returns a `cugtest` object containing the estimated 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). 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.
`pgreq`	The proportion of draws which were greater than or equal to the observed GLI value.
`pleeq`	The proportion of draws which were less than or equal to the observed GLI value.

Note

This function currently conditions only on expected statistics, and is somewhat cumbersome. `cug.test` is now recommended for univariate CUG tests (and will eventually supplant this function).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Anderson, B.S.; Butts, C.T.; and Carley, K.M. (1999). "The Interaction of Size and Density with Graph-Level Indices." *Social Networks*, 21(3), 239-267.

See Also

`cug.test`, `qaptest`, `gliop`

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)
#Now, let's try conditioning on density as well.
cug<-cugtest(dat,gcor)
summary(cug)
```

`cutpoints` *Identify the Cutpoints of a Graph or Digraph*

Description

`cutpoints` identifies the cutpoints of an input graph. Depending on `mode`, either a directed or undirected notion of "cutpoint" can be used.

Usage

```
cutpoints(dat, mode = "digraph", connected = c("strong","weak","recursive"),
    return.indicator = FALSE)
```

Arguments

`dat`	one or more input graphs.
`mode`	`"digraph"` for directed graphs, or `"graph"` for undirected graphs.
`connected`	string indicating the type of connectedness rule to apply (only relevant where `mode=="digraph"`).
`return.indicator`	logical; should the results be returned as a logical (`TRUE`/`FALSE`) vector of indicators, rather than as a vector of vertex IDs?

Details

A *cutpoint* (also known as an *articulation point* or *cut-vertex*) of an undirected graph, G is a vertex whose removal increases the number of components of G. Several generalizations to the directed case exist. Here, we define a *strong cutpoint* of directed graph G to be a vertex whose removal increases the number of strongly connected components of G (see `component.dist`). Likewise, *weak* and *recursive* cutpoints of G are those vertices whose removal increases the number of weak or recursive cutpoints (respectively). By default, strong cutpoints are used; alternatives may be selected via the `connected` argument.

Cutpoints are of particular interest when seeking to identify critical positions in flow networks, since their removal by definition alters the connectivity properties of the graph. In this context, cutpoint status can be thought of as a primitive form of centrality (with some similarities to `betweenness`).

Cutpoint computation is significantly faster for the undirected case (and for the weak/recursive cases) than for the strong directed case. While calling `cutpoints` with `mode="digraph"` on an undirected graph will give the same answer as `mode="graph"`, it is thus to one's advantage to use the latter form. Do not, however, employ `mode="graph"` with directed data, unless you enjoy unpredictable behavior.

Value

A vector of cutpoints (if `return.indicator==FALSE`), or else a logical vector indicating cutpoint status for each vertex.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Berge, Claude. (1966). *The Theory of Graphs.* New York: John Wiley and Sons.

See Also

`component.dist`, `bicomponent.dist`, `betweenness`

Examples

```
#Generate some sparse random graph
gd<-rgraph(25,tp=1.5/24)                          #Directed
gu<-rgraph(25,tp=1.5/24,mode="graph")             #Undirected

#Calculate the cutpoints (as an indicator vector)
cpu<-cutpoints(gu,mode="graph",return.indicator=TRUE)
cpd<-cutpoints(gd,return.indicator=TRUE)

#Plot the result
gplot(gu,gmode="graph",vertex.col=2+cpu)
gplot(gd,vertex.col=2+cpd)

#Repeat with alternate connectivity modes
cpdw<-cutpoints(gd,connected="weak",return.indicator=TRUE)
```

```
cpdr<-cutpoints(gd,connected="recursive",return.indicator=TRUE)

#Visualize the difference
gplot(gd,vertex.col=2+cpdw)
gplot(gd,vertex.col=2+cpdr)
```

---

degree *Compute the Degree Centrality Scores of Network Positions*

Description

Degree takes one or more graphs (dat) and returns the degree centralities of positions (selected by nodes) within the graphs indicated by g. 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

```
degree(dat, g=1, nodes=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, cmode="freeman", rescale=FALSE, ignore.eval=FALSE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.
nodes	vector 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 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.
ignore.eval	logical; should edge values be ignored when computing degree scores?

Details

Degree centrality is the social networker's term for various permutations of the graph theoretic notion of vertex degree: for unvalued graphs, 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.

When `ignore.eval==FALSE`, `degree` weights edges by their values where supplied. `ignore.eval==TRUE` ensures an unweighted degree score (independent of input). Setting `gmode=="graph"` forces behavior equivalent to `cmode=="indegree"` (i.e., each edge is counted only once); to obtain a total degree score for an undirected graph in which both in- and out-neighborhoods are counted separately, simply use `gmode=="digraph"`.

Value

A vector, matrix, or list containing the degree scores (depending on the number and size of the input graphs).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Freeman, L.C. (1979). "Centrality in Social Networks I: Conceptual Clarification." *Social Networks*, 1, 215-239.

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 graphs, with the diagonal entries removed/replaced as indicated.

Usage

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

Arguments

`dat`	one or more graphs.
`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 collection of adjacency matrices/`network` objects at once.

Value

The updated graphs.

Author(s)

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

See Also

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

Examples

```
#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=NULL)
```

Arguments

dat	one or more graphs.
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

Holland, P.W. and Leinhardt, S. (1970). "A Method for Detecting Structure in Sociometric Data." *American Journal of Sociology*, 76, 492-513.

Wasserman, S., and Faust, K. (1994). "Social Network Analysis: Methods and Applications." Cambridge: Cambridge University Press.

See Also

mutuality, grecip, rguman triad.census, kcycle.census, kpath.census

Examples

```
#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)))
```

---

efficiency   *Compute Graph Efficiency Scores*

Description

`efficiency` takes one or more graphs (`dat`) and returns the Krackhardt efficiency scores for the graphs selected by `g`.

Usage

```
efficiency(dat, g=NULL, diag=FALSE)
```

Arguments

`dat`	one or more graphs.
`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 $G = \cup_{i=1}^{n} G_i$ 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.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Krackhardt, David. (1994). "Graph Theoretical Dimensions of Informal Organizations." In K. M. Carley and M. J. Prietula (Eds.), *Computational Organization Theory*, 89-111. Hillsdale, NJ: Lawrence Erlbaum and Associates.

See Also

connectedness, efficiency, hierarchy, lubness, gden

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)))
```

`ego.extract` *Extract Egocentric Networks from Complete Network Data*

Description

`ego.extract` takes one or more input graphs (`dat`) and returns a list containing the egocentric networks centered on vertices named in ego, using adjacency rule *neighborhood* to define inclusion.

Usage

```
ego.extract(dat, ego = NULL, neighborhood = c("combined", "in",
    "out"))
```

Arguments

`dat`	one or more graphs.
`ego`	a vector of vertex IDs, or `NULL` if all are to be selected.
`neighborhood`	the neighborhood to use.

Details

The egocentric network (or "ego net") of vertex v in graph G is defined as $G[v \cup N(v)]$ (i.e., the subgraph of G induced by v and its neighborhood). The neighborhood employed by `ego.extract` is selected by the eponymous argument: `"in"` selects in-neighbors, `"out"` selects out-neighbors, and `"combined"` selects all neighbors. In the event that one of the vertices selected by ego has no qualifying neighbors, `ego.extract` will return a degenerate (1 by 1) adjacency matrix containing that individual's diagonal entry.

Vertices within the returned matrices are maintained in their original order, save for ego (who is always listed first). The ego nets themselves are returned in the order specified in the ego parameter (or their vertex order, if no value was specified).

`ego.extract` is useful for finding local properties associated with particular vertices. To compute functions of neighbors' covariates, see `gapply`.

Value

A list containing the adjacency matrices for the ego nets of each vertex in ego.

Author(s)

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

References

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

`gapply`

Examples

```
#Generate a sample network
g<-rgraph(10,tp=1.5/9)

#Extract some ego nets
g.in<-ego.extract(g,neighborhood="in")
g.out<-ego.extract(g,neighborhood="out")
g.comb<-ego.extract(g,neighborhood="in")

#View some networks
g.comb

#Compare ego net size with degree
all(sapply(g.in,NROW)==degree(g,cmode="indegree")+1)    #TRUE
all(sapply(g.out,NROW)==degree(g,cmode="outdegree")+1)  #TRUE
all(sapply(g.comb,NROW)==degree(g)/2+1)        #Usually FALSE!

#Calculate egocentric network density
ego.size<-sapply(g.comb,NROW)
```

```
if(any(ego.size>2))
  sapply(g.comb[ego.size>2],function(x){gden(x[-1,-1])})
```

---

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=NULL, equiv.dist=NULL, equiv.fun="sedist",
   method="hamming", mode="digraph", diag=FALSE,
   cluster.method="complete", glabels=NULL, plabels=NULL, ...)
```

Arguments

dat	one or more graphs.
g	the elements of dat to use in clustering the vertices; by default, all structures are used.
equiv.dist	a matrix of distances, by which vertices should be clustered. (Overrides equiv.fun.)
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.

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.fun (unless distances are supplied in 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 <buttsc@uci.edu>

References

Breiger, R.L.; Boorman, S.A.; and Arabie, P. (1975). "An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling." *Journal of Mathematical Psychology*, 12, 328-383.

Burt, R.S. (1976). "Positions in Networks." *Social Forces*, 55, 93-122.

Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

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 <`buttsc@uci.edu`>

References

Anderson, C.; Wasserman, S.; and Crouch, B. (1999). “A p* Primer: Logit Models for Social Networks. *Social Networks,* 21,37-66.

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 one or more graphs (dat) and returns the eigenvector centralities of positions (selected by nodes) within the graphs indicated by g. 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=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, rescale=FALSE, ignore.eval=FALSE, tol=1e-10,
    maxiter=1e5, use.eigen=FALSE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.
nodes	vector 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.
ignore.eval	logical; should edge values be ignored?
tol	convergence tolerance for the eigenvector computation.
maxiter	maximum iterations for eigenvector calculation.
use.eigen	logical; should we use R's eigen routine instead of the (faster but less robust) internal method?

Details

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 $S\Lambda S^{-1}$ decomposition).)

By default, a sparse-graph power method is used to obtain the principal eigenvector. This procedure scales well, but may not converge in some cases. In the event that the convergence objective set by `tol` is not obtained, `evcent` will return a warning message. Correctives in this case include increasing `maxiter`, or setting `use.eigen` to `TRUE`. The latter will cause `evcent` to use R's standard `eigen` method to calculate the principal eigenvector; this is far slower for sparse graphs, but is also more robust.

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

Value

A vector, matrix, or list containing the centrality scores (depending on the number and size of the input graphs).

WARNING

`evcent` 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 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).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Bonacich, P. (1987). "Power and Centrality: A Family of Measures." *American Journal of Sociology*, 92, 1170-1182.

Katz, L. (1953). "A New Status Index Derived from Sociometric Analysis." *Psychometrika*, 18, 39-43.

See Also

`centralization`, `bonpow`

Examples

```
#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 one or more valued adjacency matrices (possibly derived from observed interaction "events"), `event2dichot` returns dichotomized equivalents.

Usage

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

Arguments

`m`	one or more (valued) input graphs.
`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 (or matrices)

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

Examples

```
#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)
```

---

flowbet *Calculate Flow Betweenness Scores of Network Positions*

Description

flowbet takes one or more graphs (dat) and returns the flow betweenness scores of positions (selected by nodes) within the graphs indicated by g. Depending on the specified mode, flow 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

```
flowbet(dat, g = 1, nodes = NULL, gmode = "digraph", diag = FALSE,
    tmaxdev = FALSE, cmode = "rawflow", rescale = FALSE,
    ignore.eval = FALSE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.

`nodes`	vector 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 (with flows summed over directed dyads); `graph` indicates that edges are undirected (with only undirected pairs considered). `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`	one of `rawflow`, `normflow`, or `fracflow` (see below).
`rescale`	if true, centrality scores are rescaled such that they sum to 1.
`ignore.eval`	logical; ignore edge values when computing maximum flow (alternately, edge values will be assumed to carry capacity information)?

Details

The ("raw," or unnormalized) flow betweenness of a vertex, $v \in V(G)$, is defined by Freeman et al. (1991) as

$$C_F(v) = \sum_{i,j:i \neq j, i \neq v, j \neq v} \left(f(i,j,G) - f(i,j,G \setminus v)\right),$$

where $f(i,j,G)$ is the maximum flow from i to j within G (under the assumption of infinite vertex capacities, finite edge capacities, and non-simultaneity of pairwise flows). Intuitively, unnormalized flow betweenness is simply the total maximum flow (aggregated across all pairs of third parties) mediated by v.

The above flow betweenness measure is computed by `flowbet` when `cmode=="rawflow"`. In some cases, it may be desirable to normalize the raw flow betweenness by the total maximum flow among third parties (including v); this leads to the following normalized flow betweenness measure:

$$C'_F(v) = \frac{\sum_{i,j:i \neq j, i \neq v, j \neq v} \left(f(i,j,G) - f(i,j,G \setminus v)\right)}{\sum_{i,j:i \neq j, i \neq v, j \neq v} f(i,j,G)}.$$

This variant can be selected by setting `cmode=="normflow"`.

Finally, it may be noted that the above normalization (from Freeman et al. (1991)) is rather different from that used in the definition of shortest-path betweenness, which normalizes within (rather than across) third-party dyads. A third flow betweenness variant has been suggested by Koschutzki et al. (2005) based on a normalization of this type:

$$C''_F(v) = \sum_{i,j:i \neq j, i \neq v, j \neq v} \frac{\left(f(i,j,G) - f(i,j,G \setminus v)\right)}{f(i,j,G)}$$

where 0/0 flow ratios are treated as 0 (as in shortest-path betweenness). Setting `cmode=="fracflow"` selects this variant.

Value

A vector of centrality scores.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Freeman, L.C.; Borgatti, S.P.; and White, D.R. (1991). "Centrality in Valued Graphs: A Measure of Betweenness Based on Network Flow." *Social Networks*, 13(2), 141-154.

Koschutzki, D.; Lehmann, K.A.; Peeters, L.; Richter, S.; Tenfelde-Podehl, D.; Zlotowski, O. (2005). "Centrality Indices." In U. Brandes and T. Erlebach (eds.), *Network Analysis: Methodological Foundations.* Berlin: Springer.

See Also

betweenness, maxflow

Examples

```
g<-rgraph(10)                              #Draw a random graph
flowbet(g)                                 #Raw flow betweenness
flowbet(g,cmode="normflow")                #Normalized flow betweenness

g<-g*matrix(rpois(100,4),10,10)            #Add capacity constraints
flowbet(g)                                 #Note the difference!
```

---

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

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

Arguments

X	one or more input graphs.
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, gapply first identifies all members of the relevant neighborhood (as determined by MARGIN and distance) and pulls the rows of STATS associated with each. 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 FUN and STATS, a wide range of functionality can be obtained.

Value

The result of the iterated application of FUN to each vertex neighborhood's STATS.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

apply, sapply

Examples

```
#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="indegree"))
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)
```

gclust.boxstats *Plot Statistics Associated with Graph Clusters*

Description

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

Usage

```
gclust.boxstats(h, k, meas, ...)
```

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 <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

See Also

gclust.centralgraph, gdist.plotdiff, gdist.plotstats

Examples

```
#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 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, dat, ...)
```

Arguments

`h`	an `hclust` object, based on a graph stack in `dat`.
`k`	the number of groups to evaluate.
`dat`	one or more 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

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

See Also

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

Examples

```
#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

```
gcor(dat, dat2=NULL, g1=NULL, g2=NULL, diag=FALSE, mode="digraph")
```

Arguments

`dat`	one or more input graphs.
`dat2`	optionally, a second stack of graphs.
`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 (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

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

(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 `gcor` computes only the correlation between *uniquely labeled* graphs. For the more general case, `gscor` is recommended.

Value

A graph correlation matrix

Note

The `gcor` routine is really just a front-end to the standard `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 `cugtest` and `qaptest`. For multivariate correlations among graph sets, try `netcancor`.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS Working Paper, Carnegie Mellon University.

Krackhardt, D. (1987). "QAP Partialling as a Test of Spuriousness." *Social Networks*, 9, 171-86

See Also

gscor, gcov, gscov

Examples

```
#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)
```

---

gcov *Find the Covariance(s) Between Two or More Labeled Graphs*

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=NULL, g2=NULL, diag=FALSE, mode="digraph")
```

Arguments

dat	one or more input graphs.
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

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

(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.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS Working Paper, Carnegie Mellon University.

See Also

gscov, gcor, gscor

Examples

```
#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 covariance matrix
gcov(g)
```

---

gden *Find the Density of a Graph*

Description

gden computes the density of the graphs indicated by g in collection dat, adjusting for the type of graph in question.

Usage

```
gden(dat, g=NULL, diag=FALSE, mode="digraph", ignore.eval=FALSE)
```

Arguments

`dat`	one or more input graphs.
`g`	integer indicating the index of the graphs for which the density is to be calculated (or a vector thereof). If `g==NULL` (the default), density is calculated for all graphs in `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. `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.
`ignore.eval`	logical; should edge values be ignored when calculating density?

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 when `ignore.eval==FALSE`. The number of possible ties is determined by the graph type (and by `diag`) in the usual fashion.

Where missing data is present, it is removed prior to calculation. The density/graph mean is thus taken relative to the observed portion of the graph.

Value

The graph density

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

Examples

```
#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

```
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 <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

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 "normalize" 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 properties 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.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

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, count.paths=TRUE, predecessors=FALSE,
    ignore.eval=TRUE, na.omit=TRUE)
```

Arguments

`dat`	one or more input graphs.
`inf.replace`	the value to use for geodesic distances between disconnected nodes; by default, this is equal `Inf`.
`count.paths`	logical; should a count of geodesics be included in the returned object?
`predecessors`	logical; should a predecessor list be included in the returned object?
`ignore.eval`	logical; should edge values be ignored when computing geodesics?
`na.omit`	logical; should `NA`-valued edges be removed?

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 infinite 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.

By default, geodist ignores edge values (except for NAed edges, which are dropped when na.omit==TRUE). Setting ignore.eval=FALSE will change this behavior, with edge values being interpreted as distances; where edge values reflect proximity or tie strength, transformation may be necessary. Edge values should also be non-negative. Because the valued-case algorithm is significantly slower than the unvalued-case algorithm, ignore.eval should be set to TRUE wherever possible.

Value

A list containing:

counts	If count.paths==TRUE, a matrix containing the number of geodesics between each pair of vertices
gdist	A matrix containing the geodesic distances between each pair of vertices
predecessors	If predecessors, a list whose ith element is a list of vectors, the jth of which contains the intervening vertices on all shortest paths from i to j

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Brandes, U. (2000). "Faster Evaluation of Shortest-Path Based Centrality Indices." *Konstanzer Schriften in Mathematik und Informatik*, 120.

West, D.B. (1996). *Introduction to Graph Theory.* Upper Saddle River, N.J.: Prentice Hall.

See Also

component.dist, components

Examples

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

#Examine the number of geodesics
gd$counts

#Examine the geodesic distances
gd$gdist
```

`gilschmidt` *Compute the Gil-Schmidt Power Index*

Description

`gilschmidt` computes the Gil-Schmidt Power Index for all nodes in `dat`, with or without normalization.

Usage

```
gilschmidt(dat, g = 1, nodes = NULL, gmode = "digraph", diag = FALSE,
    tmaxdev = FALSE, normalize = TRUE)
```

Arguments

`dat`	one or more input graphs (for best performance, sna edgelists or network objects are suggested).
`g`	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). 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. (This has no effect on this index, but is included for compatibility with `centralization`.
`tmaxdev`	boolean indicating whether or not the theoretical maximum absolute deviation from the maximum nodal centrality should be returned. By default, `tmaxdev==FALSE`.
`normalize`	logical; should the index scores be normalized?

Details

For graph $G = (V, E)$, let $R(v, G)$ be the set of vertices reachable by v in $V \setminus v$. Then the Gil-Schmidt power index is defined as

$$C_{GS}(v) = \frac{\sum_{i \in R(v,G)} \frac{1}{d(v,i)}}{|R(v, G)|}.$$

where $d(v, i)$ is the geodesic distance from v to i in G; the index is taken to be 0 for isolates. The measure takes a value of 1 when v is adjacent to all reachable vertices, and approaches 0 as the distance from v to each vertex approaches infinity. (For finite $N = |V|$, the minimum value is 0 if v is an isolate, and otherwise $1/(N - 1)$.)

If `normalize=FALSE` is selected, then normalization by $|R(v, G)|$ is not performed. This measure has been proposed as a better-behaved alternative to closeness (to which it is closely related).

The `closeness` function in the sna library can also be used to compute this index.

Value

A vector of centrality scores.

Author(s)

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

References

Gil, J. and Schmidt, S. (1996). "The Origin of the Mexican Network of Power". Proceedings of the International Social Network Conference, Charleston, SC, 22-25.

Sinclair, P.A. (2009). "Network Centralization with the Gil Schmidt Power Centrality Index" *Social Networks*, 29, 81-92.

See Also

closeness, centralization

Examples

```
data(coleman)  #Load Coleman friendship network
gs<-gilschmidt(coleman,g=1:2)  #Compute the Gil-Schmidt index

#Plot G-S values in the fall, versus spring
plot(gs,xlab="Fall",ylab="Spring",main="G-S Index")
abline(0,1)
```

---

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

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

Arguments

dat	a collection of graphs.
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

OP(GFUN(dat[g1,,],...),GFUN(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

Anderson, B.S.; Butts, C.T.; and Carley, K.M. (1999). "The Interaction of Size and Density with Graph-Level Indices." *Social Networks*, 21(3), 239-267.

See Also

cugtest, qaptest

Examples

```
#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 *Two-Dimensional Visualization of Graphs*

Description

gplot produces a two-dimensional plot of graph g in collection 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 = NULL, coord = NULL, jitter = TRUE, thresh = 0,
    thresh.absval=TRUE, usearrows = TRUE, mode = "fruchtermanreingold",
    displayisolates = TRUE, interactive = FALSE, interact.bycomp = FALSE,
    xlab = NULL, ylab = NULL, xlim = NULL, ylim = NULL, pad = 0.2,
    label.pad = 0.5, displaylabels = !is.null(label), boxed.labels = FALSE,
    label.pos = 0, label.bg = "white", vertex.enclose = FALSE,
    vertex.sides = NULL, vertex.rot = 0, arrowhead.cex = 1, label.cex = 1,
    loop.cex = 1, vertex.cex = 1, edge.col = 1, label.col = 1,
    vertex.col = NULL, label.border = 1, vertex.border = 1, edge.lty = NULL,
    edge.lty.neg=2, 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 set thereof. 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 two-mode (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` (by default in absolute value - see `thresh.absval`)are displayed. By default, `thresh`=0.
`thresh.absval`	boolean; should the absolute value of edge weights be used when thresholding? (Defaults to TRUE; setting to FALSE leads to thresholding by signed weights.)
`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?
`interact.bycomp`	boolean; if `interactive==TRUE`, should all vertices in the component be moved?

`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.enclose`	boolean; should vertices be enclosed within circles? (Can increase legibility for polygonal vertices.)
`vertex.sides`	number of polygon sides for vertices; may be given as a vector, if vertices are to be of different types. By default, 50 sides are used (or 50 and 4, for two-mode data).
`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. By default, red is used (or red and blue, for two-mode data).
`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 (positive weight) edges; may be given as a vector or adjacency matrix, if edges are to have different line types.
`edge.lty.neg`	line type for negative weight edges, if any; may be given as per `edge.lty`.

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.
edge.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 or adjacency matrix, if edges 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 network is converted to bipartite form prior to computing coordinates (if not in that form already). vertex.col or other settings may be used to differentiate row and column vertices – by default, row vertices are drawn as red circles, and column vertices are rendered as blue squares. 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. If interact.bycomp==TRUE as well, the vertex and all other vertices in the same component as that vertex are moved together.

Value

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

Author(s)

Carter T. Butts <buttsc@uci.edu>

Alex Montgomery <ahm@reed.edu>

References

Wasserman, S. and Faust, K. (1994) *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

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")
```

gplot.layout *Vertex Layout Functions for gplot*

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.mds(d, layout.par)
gplot.layout.princoord(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. Larger values take longer, but will provide a more refined layout. (Defaults to 500.)

layout.par\$max.delta Sets the maximum change in position for any given iteration. (Defaults to `n`.)

layout.par\$area Sets the "area" parameter for the F-R algorithm. (Defaults to `n^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*log(n)`.)

layout.par\$ncell To speed calculations on large graphs, the plot region is divided at each iteration into `ncell` by `ncell` "cells", which are used to define neighborhoods for force calculation. Moderate numbers of cells result in fastest performance; too few cells (down to 1, which produces "pure" F-R results) can yield odd layouts, while too many will result in long layout times. (Defaults to `n^0.4`.)

layout.par\$cell.jitter Jitter factor (in units of cell width) used in assigning vertices to cells. Small values may generate "grid-like" anomalies for graphs with many isolates. (Defaults to `0.5`.)

layout.par\$cell.pointpointrad Squared "radius" (in units of cells) such that exact point interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart. Higher values approximate the true F-R solution, but increase computational cost. (Defaults to `0`.)

layout.par\$cell.pointcellrad Squared "radius" (in units of cells) such that approximate point/cell interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart (and not within the point/point radius). Higher values provide somewhat better approximations to the true F-R solution at slightly increased computational cost. (Defaults to `18`.)

layout.par\$cell.cellcellrad Squared "radius" (in units of cells) such that approximate cell/cell interaction calculations are used for all vertices belonging to any two cells less than or equal to this distance apart (and not within the point/point or point/cell radii). Higher values provide somewhat better approximations to the true F-R solution at slightly increased computational cost. Note that cells beyond this radius (if any) do not interact, save through edge attraction. (Defaults to `ncell^2`.)

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\$initemp 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 `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: circrand, 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

Brandes, U.; Kenis, P.; and Wagner, D. (2003). “Communicating Centrality in Policy Network Drawings.” *IEEE Transactions on Visualization and Computer Graphics,* 9(2):241-253.

Fruchterman, T.M.J. and Reingold, E.M. (1991). “Graph Drawing by Force-directed Placement.” *Software - Practice and Experience,* 21(11):1129-1164.

Kamada, T. and Kawai, S. (1989). “An Algorithm for Drawing General Undirected Graphs.” *Information Processing Letters,* 31(1):7-15.

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

```
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, polygon

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))
```

`gplot.target` *Display a Graph in Target Diagram Form*

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

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

Arguments

`dat`	an input graph.
`x`	a vector of vertex properties to be plotted (must match the dimensions of `dat`).
`circ.rad`	radii at which to draw reference circles.
`circ.col`	reference circle color.
`circ.lwd`	reference circle line width.
`circ.lty`	reference circle line type.
`circ.lab`	boolean; should circle labels be displayed?
`circ.lab.cex`	expansion factor for circle labels.
`circ.lab.theta`	angle at which to draw circle labels.
`circ.lab.col`	color for circle labels.
`circ.lab.digits`	digits to display for circle labels.
`circ.lab.offset`	offset for circle labels.
`periph.outside`	boolean; should "peripheral" vertices be drawn together beyond the normal vertex radius?
`periph.outside.offset`	radius at which "peripheral" vertices should be drawn if `periph.outside==TRUE`.
`...`	additional arguments to `gplot`.

Details

gplot.target is a front-end to 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 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 gplot.target is handled by gplot.layout.target; additional parameters are specied on the associated manual page. Standard arguments may be passed to gplot, as well.

Value

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

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Brandes, U.; Kenis, P.; and Wagner, D. (2003). "Communicating Centrality in Policy Network Drawings." *IEEE Transactions on Visualization and Computer Graphics,* 9(2):241-253.

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

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`gplot`, `polygon`

Examples

```
#Open a plot window, and place some vertices
plot(0,0,type="n",xlim=c(-1.5,1.5),ylim=c(-1.5,1.5),asp=1)
gplot.vertex(cos((1:10)/10*2*pi),sin((1:10)/10*2*pi),col=1:10,
    sides=3:12,radius=0.1)
```

gplot3d *Three-Dimensional Visualization of Graphs*

Description

gplot3d produces a three-dimensional plot of graph g in set 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 = NULL, coord = NULL, jitter = TRUE, thresh = 0,
    mode = "fruchtermanreingold", displayisolates = TRUE,
    displaylabels = !missing(label), xlab = NULL, ylab = NULL,
    zlab = NULL, vertex.radius = NULL, absolute.radius = FALSE,
    label.col = "gray50", edge.col = "black", vertex.col = NULL,
    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 set thereof. 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)x3 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.
`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. By default, red is used (or red and blue, if `gmode=="twomode"`).
`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.
`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 graph is converted to bipartite form prior to computing coordinates (assuming it is not in this form already). It may be desirable to use parameters such as `vertex.col` to differentiate row and column vertices; by default, row vertices are colored red, and column vertices blue.

Value

A three-column matrix containing vertex coordinates

Requires

`rgl`

Author(s)

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

References

Wasserman, S. and Faust, K. (1994) *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

gplot, gplot3d.layout, rgl

Examples

```
## 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,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 NROW(d).)

layout.par\$volume Sets the "volume" parameter for the F-R algorithm. (Defaults to 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 volume*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 NROW(d)/4.)

layout.par\$initemp 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)^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(NROW(d)).)

layout.par\$seed.coord A three-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 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 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 <buttsc@uci.edu>

References

Fruchterman, T.M.J. and Reingold, E.M. (1991). “Graph Drawing by Force-directed Placement.” *Software - Practice and Experience,* 21(11):1129-1164.

Kamada, T. and Kawai, S. (1989). “An Algorithm for Drawing General Undirected Graphs.” *Information Processing Letters,* 31(1):7-15.

See Also

gplot3d, gplot, gplot.layout, cmdscale, eigen

---

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 noticable as spike-like structures which protrude from vertices. Eventually, something more attractice will be produced by this routine.

Value

None.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

gplot3d.arrow, gplot3d, rgl-package

---

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

Description

graphcent takes one or more graphs (dat) and returns the Harary graph centralities of positions (selected by nodes) within the graphs indicated by g. 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

```
graphcent(dat, g=1, nodes=NULL, gmode="digraph", diag=FALSE,
    tmaxdev=FALSE, cmode="directed", geodist.precomp=NULL,
    rescale=FALSE, ignore.eval)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). 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.
ignore.eval	logical; should edge values be ignored when calculating geodesics?

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, matrix, or list containing the centrality scores (depending on the number and size of the input graphs).

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

Hage, P. and Harary, F. (1995). “Eccentricity and Centrality in Networks.” *Social Networks*, 17:57-63.

See Also

`centralization`

Examples

```
g<-rgraph(10)     #Draw a random graph with 10 members
graphcent(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

```
grecip(dat, g = NULL, measure = c("dyadic", "dyadic.nonnull",
    "edgewise", "edgewise.lrr", "correlation"))
```

Arguments

`dat`	one or more input graphs.
`g`	a vector indicating which graphs to evaluate (optional).
`measure`	one of `"dyadic"` (default), `"dyadic.nonnull"`, `"edgewise"`, `"edgewise.lrr"`, or `"correlation"`.

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. (`dyadic.nonnull` returns the ratio of mutuals to non-null dyads.) 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"`. Setting `measure="edgewise.lrr"` returns the log of the ratio of the edgewise reciprocity to the density; this is measure (called r_4 by Butts (2008)) can be interpreted as the relative log-odds of an edge given a reciprocation, versus the baseline probability of an edge. Finally, `measure="correlation"` returns the correlation between within-dyad edge values, where this is defined by

$$\frac{2\sum_{\{i,j\}}(Y_{ij}-\mu_G)(Y_{ji}-\mu_G)}{(2N_d-1)\sigma^2_G}$$

with Y being the graph adjacency matrix, μ_G being the mean non-loop edge value, σ^2_G being the variance of non-loop edge values, and N_d being the number of dyads. (Note that this quantity is unaffected by dyad orientation.) The correlation measure may be interpreted as the net tendency for edges of similar relative value (with respect to the mean edge value) to occur within the same dyads. For dichotomous data, adjacencies are interpreted as having values of 0 (no edge present) or 1 (edge present), but edge values are used where supplied. In cases where all edge values are identical (e.g., the complete or empty graph), the correlation reciprocity is taken to be 1 by definition.

Note that `grecip` calculates values based on non-missing data; dyads containing missing data are removed from consideration when calculating reciprocity scores (except for the correlation measure, which uses non-missing edges within missing dyads when calculating the graph mean and variance).

Value

The graph reciprocity value(s)

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

Butts, C.T. (2008). "Social Networks: A Methodological Introduction." *Asian Journal of Social Psychology,* 11(1), 13-41.

See Also

`mutuality`, `symmetrize`

Examples

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

---

gscor *Find the Structural Correlations Between Two or More Graphs*

Description

`gscor` finds the product-moment structural correlation 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

```
gscor(dat, dat2=NULL, g1=NULL, g2=NULL,  diag=FALSE,
    mode="digraph", method="anneal", reps=1000, prob.init=0.9,
    prob.decay=0.85, freeze.time=25, full.neighborhood=TRUE,
    exchange.list=0)
```

Arguments

`dat` a stack of input graphs.

`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 correlation coefficient between two graphs G and H is defined as

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

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 hypothesis tests should not be employed with structural correlations, 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 correlation 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>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS Working Paper, Carnegie Mellon University.

See Also

gscov, gcor, gcov

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 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)
```

---

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

```
gscov(dat, dat2=NULL, g1=NULL, g2=NULL, diag=FALSE, mode="digraph",
    method="anneal", reps=1000, prob.init=0.9, prob.decay=0.85,
    freeze.time=25, full.neighborhood=TRUE, exchange.list=0)
```

Arguments

`dat`	one or more input graphs.
`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`	dhould 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

$$scov\left(G, H \left| L_G, L_H\right.\right) = \max_{L_G, L_H} cov(\ell(G), \ell(H))$$

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 exchangeabiliy 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>

References

Butts, C.T., and Carley, K.M. (2001). “Multivariate Methods for Interstructural Analysis.” CASOS Working Paper, Carnegie Mellon University.

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)
```

---

gt *Transpose an Input Graph*

Description

`gt` returns the graph transpose of its input. For an adjacency matrix, this is the same as using `t`; however, this function is also applicable to sna edgelists (which cannot be transposed in the usual fashion). Code written using `gt` instead of `t` is thus guaranteed to be safe for either form of input.

Usage

```
gt(x, return.as.edgelist = FALSE)
```

Arguments

`x`	one or more graphs.
`return.as.edgelist`	logical; should the result be returned in sna edgelist form?

Details

The transpose of a (di)graph, $G = (V, E)$, is the graph $G = (V, E')$ where $E' = \{(j, i) : (i, j) \in E\}$. This is simply the graph formed by reversing the sense of the edges.

Value

The transposed graph(s).

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

symmetrize, t

Examples

```
#Create a graph....
g<-rgraph(5)
g

#Transpose it
gt(g)
gt(g)==t(g)                  #For adjacency matrices, same as t(g)

#Now, see both versions in edgelist form
as.edgelist.sna(g)
gt(g,return.as.edgelist=TRUE)
```

---

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", "rank", "correlation"),
    use.adjacency = TRUE)
```

Arguments

dat	a collection of input graphs.
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", "strongcensus", "rank", or "correlation".

use.adjacency — logical; should adjacency matrices (versus sparse graph methods) be used in the transitivity computation?

Details

Transitivity is a triadic, algebraic structural constraint. In its weak form, the transitive constraint corresponds to $a \to b \to c \Rightarrow a \to c$. In the corresponding strong form, the constraint is $a \to b \to c \Leftrightarrow a \to c$. (Note that the weak form is that most commonly employed.) Where measure=="weak", the fraction of potentially intransitive triads obeying the weak condition is returned. With the measure=="weakcensus" 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).

Note that where missing values prevent the assessment of whether a triple is transitive, that triple is omitted.

Generalizations of transitivity to valued graphs are numerous. The above strong and weak forms ignore edge values, treating any non-zero edge as present. Two additional notions of transitivity are also supported valued data. The "rank" condition treads an (i, j, k) triple as transitive if the value of the (i, k) directed dyad is greater than or equal to the minimum of the values of the (i, j) and (j, k) dyads. The "correlation" option implements the correlation transitivity of David Dekker, which is defined as the matrix correlation of the valued adjacency matrix A with its second power (i.e., A^2), omitting diagonal entries where inapplicable.

Note that the base forms of transitivity can be calculated using either matrix multiplication or sparse graph methods. For very large, sparse graphs, the sparse graph method (which can be forced by use.adjacency=FALSE) may be preferred. The latter provides much better scaling, but is significantly slower for networks of typical size due to the overhead involved (and R's highly optimized matrix operations). Where use.adjacency is set to TRUE, gtrans will attempt some simple heuristics to determine if the edgelist method should be used instead (and will do so if indicated). These heuristics depend on recognition of the input data type, and hence may behave slightly differently depending on the form in which dat is given. Note that the rank measure can at present be calculated only via sparse graph methods, and the correlation measure only by adjacency matrices. For these measures, the use.adjacency argument is ignored.

Value

A vector of transitivity scores

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Holland, P.W., and Leinhardt, S. (1972). "Some Evidence on the Transitivity of Positive Interpersonal Sentiment." *American Journal of Sociology,* 72, 1205-1209.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

triad.classify, cugtest

Examples

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

#Find transitivity scores
gtrans(g)
```

---

gvectorize *Vectorization of Adjacency Matrices*

Description

gvectorize takes an input graph set and converts it into a corresponding number of vectors by row concatenation.

Usage

```
gvectorize(mats, mode="digraph", diag=FALSE, censor.as.na=TRUE)
```

Arguments

mats	one or more input graphs.
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 set 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=NULL, g2=NULL, normalize=FALSE,
    diag=FALSE, mode="digraph")
```

Arguments

dat	a stack of input graphs.
dat2	a second graph stack (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 $|\{e : (e \in E(G_1), e \notin E(G_2)) \wedge (e \notin E(G_1), e \in E(G_2))\}|$. 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 <buttsc@uci.edu>

References

Banks, D., and Carley, K.M. (1994). "Metric Inference for Social Networks." *Journal of Classification*, 11(1), 121-49.

Butts, C.T. and Carley, K.M. (2005). "Some Simple Algorithms for Structural Comparison." *Computational and Mathematical Organization Theory,* 11(4), 291-305.

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS Working Paper, Carnegie Mellon University.

Hamming, R.W. (1950). "Error Detecting and Error Correcting Codes." *Bell System Technical Journal,* 29, 147-160.

See Also

sdmat, structdist

Examples

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

#Find the Hamming distances
hdist(g)
```

`hierarchy` *Compute Graph Hierarchy Scores*

Description

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

Usage

```
hierarchy(dat, g=NULL, measure=c("reciprocity", "krackhardt"))
```

Arguments

`dat`	a stack of input graphs.
`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.

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

Krackhardt, David. (1994). "Graph Theoretical Dimensions of Informal Organizations." In K. M. Carley and M. J. Prietula (Eds.), *Computational Organization Theory*, 89-111. Hillsdale, NJ: Lawrence Erlbaum and Associates.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

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 one or more graphs (dat) and returns the information centralities of positions (selected by nodes) within the graphs indicated by g. 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=NULL, gmode="digraph", diag=FALSE,
    cmode="weak", tmaxdev=FALSE, rescale=FALSE,tol=1e-20)
```

Arguments

`dat`	one or more input graphs.
`g`	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). 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 $\mathbf{A}$. From this, we remove all isolates (whose information centralities are zero in any event) and proceed to create the weighted connection matrix

$$\mathbf{C} = \mathbf{B}^{-1}$$

where $\mathbf{B}$ is a pseudo-adjacency matrix formed by replacing the diagonal of $1-\mathbf{A}$ with one plus each actor's degree. Given the above, let T be the trace of $\mathbf{C}$ with sum S_T, and let S_R be an arbitrary row sum (all rows of $\mathbf{C}$ 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 $\mathbf{B}$ matrix poses problems of its own; as with all such measures, caution is advised on disconnected or degenerate structures.

Value

A vector, matrix, or list containing the centrality scores (depending on the number and size of the input graphs).

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

Stephenson, K., and Zelen, M. (1989). "Rethinking Centrality: Methods and Applications." *Social Networks*, 11, 1-37.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

evcent, bonpow, closeness, graphcent, centralization

Examples

```
#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

```
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

Butts, C.T. and Pixley, J.E. (2004). "A Structural Approach to the Representation of Life History Data." *Journal of Mathematical Sociology*, 28(2), 81-124.

West, D.B. (1996). *Introduction to Graph Theory*. Upper Saddle River, NJ: Prentice Hall.

---

is.connected *Is a Given Graph Connected?*

Description

Returns TRUE iff the specified graphs are connected.

Usage

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

Arguments

g	one or more input graphs.
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 the elements of g are 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

West, D.B. (1996). *Introduction to Graph Theory.* Upper Saddle River, N.J.: Prentice Hall.

See Also

component.dist, components

Examples

```
#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
```

`is.isolate` *Is Ego an Isolate?*

Description

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

Usage

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

Arguments

`dat`	one or more input graphs.
`ego`	index of the vertex (or a vector of vertices) to check.
`g`	which graph(s) 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 <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

West, D.B. (1996). *Introduction to Graph Theory*. Upper Saddle River, NJ: Prentice Hall.

See Also

isolates, add.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 set given by dat.

Usage

```
isolates(dat, diag=FALSE)
```

Arguments

dat	one or more input graphs.
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 <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

West, D.B. (1996). *Introduction to Graph Theory*. Upper Saddle River, NJ: Prentice Hall.

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)
```

---

kcores *Compute the k-Core Structure of a Graph*

Description

kcores calculates the k-core structure of the input network, using the centrality measure indicated in cmode.

Usage

```
kcores(dat, mode = "digraph", diag = FALSE, cmode = "freeman",
    ignore.eval = FALSE)
```

Arguments

dat	one or more (possibly valued) graphs.
mode	"digraph" for directed data, otherwise "graph".
diag	logical; should self-ties be included in the degree calculations?
cmode	the degree centrality mode to use when constructing the cores.
ignore.eval	logical; should edge values be ignored when computing degree?

Details

Let $G = (V, E)$ be a graph, and let $f(v, S, G)$ for $v \in V, S \subseteq V$ be a real-valued *vertex property function* (in the language of Batagelj and Zaversnik). Then some set $H \subseteq V$ is a *generalized k-core* for f if H is a maximal set such that $f(v, H, G) \geq k$ for all $v \in H$. Typically, f is chosen to be a degree measure with respect to S (e.g., the number of ties to vertices in S). In this case, the resulting k-cores have the intuitive property of being maximal sets such that every set member is tied (in the appropriate manner) to at least k others within the set.

Degree-based k-cores are a simple tool for identifying well-connected structures within large graphs. Let the *core number* of vertex v be the value of the highest-value core containing v. Then, intuitively, vertices with high core numbers belong to relatively well-connected sets (in the sense of sets with high minimum internal degree). It is important to note that, while a given k-core need not be connected, it is composed of subsets which are themselves well-connected; thus, the k-cores can be thought of as unions of relatively cohesive subgroups. As k-cores are nested, it is also natural to think of each k-core as representing a "slice" through a hypothetical "cohesion surface" on G. (Indeed, k-cores are often visualized in exactly this manner.)

The kcores function produces degree-based k-cores, for various degree measures (with or without edge values). The return value is the vector of core numbers for V, based on the selected degree measure. Missing (i.e., NA) edge are removed for purposes of the degree calculation.

Value

A vector containing the maximum core membership for each vertex.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Batagelj, V. and Zaversnik, M. (2002). "An $O(m)$ Algorithm for Cores Decomposition of Networks." arXiv:cs/0310049v1

Batagelj, V. and Zaversnik, M. (2002). "Generalized Cores." arXiv:cs/0202039v1

Wasserman, S. and Faust,K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

degree

Examples

```
#Generate a graph with core-periphery structure
cv<-runif(30)
g<-rgraph(30,tp=cv%o%cv)

#Compute the k-cores based on total degree
kc<-kcores(g)
kc

#Plot the result
gplot(g,vertex.col=kc)
```

---

kpath.census *Compute Path or Cycle Census Information*

Description

kpath.census and kcycle.census compute k-path or k-cycle census statistics (respectively) on one or more input graphs. In addition to aggregate counts of paths or cycles, results may be disaggregated by vertex and co-membership information may be computed.

Usage

```
kcycle.census(dat, maxlen = 3, mode = "digraph",
    tabulate.by.vertex = TRUE, cycle.comembership = c("none", "sum",
    "bylength"))

kpath.census(dat, maxlen = 3, mode = "digraph",
    tabulate.by.vertex = TRUE, path.comembership = c("none", "sum",
    "bylength"), dyadic.tabulation = c("none", "sum", "bylength"))
```

Arguments

`cycle.comembership`
: the type of cycle co-membership information to be tabulated, if any. "sum" returns a vertex by vertex matrix of cycle co-membership counts; these are disaggregated by cycle length if "bylength" is used. If "none" is given, no co-membership information is computed.

`dat`
: one or more input graphs.

`maxlen`
: the maximum path/cycle length to evaluate.

`mode`
: "digraph" for directed graphs, or "graph" for undirected graphs.

`tabulate.by.vertex`
: logical; should path or cycle incidence counts be tabulated by vertex?

`path.comembership`
: as per `cycle.comembership`, for paths rather than cycles.

`dyadic.tabulation`
: the type of dyadic path count information to be tabulated, if any. "sum" returns a vertex by vertex matrix of source/destination path counts, while "bylength" disaggregates these counts by path length. Selecting "none" disables this computation.

Details

There are several equivalent characterizations of paths and cycles, of which the following is one example. For an arbitrary graph G, a *path* is a sequence of distinct vertices $v_1, v_2, \ldots, v_n$ and included edges such that v_i is adjacent to v_{i+1} for all $i \in 1, 2, \ldots, n-1$ via the pair's included edge. (Contrast this with a *walk*, in which edges and/or vertices may be repeated.) A *cycle* is the union of a path and an edge making v_n adjacent to v_i. k-paths and k-cycles are respective paths and cycles having k edges (in the former case) or k vertices (in the latter). The above definitions may be applied in both directed and undirected contexts, by substituting the appropriate notion of adjacency. (Note that authors do not always employ the same terminology for these concepts, especially in older texts – it is wise to verify the definitions being used in any particular context.)

A *subgraph census statistic* is a function which, for any given graph and subgraph, gives the number of copies of the latter contained in the former. A collection of subgraph census statistics is referred to as a *subgraph census*; widely used examples include the dyad and triad censuses, implemented in sna by the `dyad.census` and `triad.census` functions (respectively). `kpath.census` and `kcycle.census` compute a range of census statistics related to k-paths and k-cycles, including:

- Aggregate counts of paths/cycles by length (i.e., k).

- Counts of paths/cycles to which each vertex belongs (when `tabulate.byvertex==TRUE`).
- Counts of path/cycle co-memberships, potentially disaggregated by length (when the appropriate co-membership argument is set to `bylength`).
- For `path.census`, counts of the total number of paths from each vertex to each other vertex, possibly disaggregated by length (if `dyadic.tabulation=="bylength"`).

The length of the maximum-length path/cycle to compute is given by `maxlen`. These calculations are intrinsically expensive (path/cycle computation is NP complete in the general case), and users should hence be wary when increasing `maxlen`. On the other hand, it may be possible to enumerate even long paths or cycles on a very sparse graph; scaling is approximately c^k, where k is given by `maxlen` and c is the size of the largest dense cluster.

The paths or cycles computed by this function are directed if `mode=="digraph"`, or undirected if `mode=="graph"`. Failing to set `mode` correctly may result in problematic behavior.

Value

For `kpath.census`, a list with the following elements:

`path.count` If `tabulate.byvertex==FALSE`, a vector of aggregate counts by path length. Otherwise, a matrix whose first column is a vector of aggregate path counts, and whose succeeding columns contain vectors of path counts for each vertex.

`path.comemb` If `path.comembership!="none"`, a matrix or array containing co-membership in paths by vertex pairs. If `path.comembership=="sum"`, only a matrix of co-memberships is returned; if `bylength` is used, however, co-memberships are returned in a `maxlen` by n by n array whose i, j, kth cell is the number of paths of length i containing `j` and `k`.

`paths.bydyad` If `dyadic.tabulation!="none"`, a matrix or array containing the number of paths originating at a particular vertex and terminating. If `bylength` is used, dyadic path counts are supplied via a `maxlen` by n by n array whose i, j, kth cell is the number of paths of length i starting at `j` and ending with `k`. If `sum` is used instead, only a matrix whose i, j cell contains the total number of paths from i to j is returned.

For `kcycle.census`, a similar list:

`cycle.count` If `tabulate.byvertex==FALSE`, a vector of aggregate counts by cycle length. Otherwise, a matrix whose first column is a vector of aggregate cycle counts, and whose succeeding columns contain vectors of cycle counts for each vertex.

`cycle.comemb` If `cycle.comembership!="none"`, a matrix or array containing co-membership in cycles by vertex pairs. If `cycle.comembership=="sum"`, only a matrix of co-memberships is returned; if `bylength` is used, however, co-memberships are returned in a `maxlen` by n by n array whose i, j, kth cell is the number of cycles of length i containing `j` and `k`.

Warning

The computational cost of calculating paths and cycles grows very sharply in both `maxlen` and network density. Be wary of setting `maxlen` greater than 5-6, unless you know what you are doing. Otherwise, the expected completion time for your calculation may exceed your life expectancy (and those of subsequent generations).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T. (2006). "Cycle Census Statistics for Exponential Random Graph Models." IMBS Technical Report MBS 06-05, University of California, Irvine.

West, D.B. (1996). *Introduction to Graph Theory.* Upper Saddle River, N.J.: Prentice Hall.

See Also

dyad.census, triad.census, clique.census, geodist

Examples

```
g<-rgraph(20,tp=1.5/19)

#Obtain paths by vertex, with dyadic path counts
pc<-kpath.census(g,maxlen=5,dyadic.tabulation="sum")
pc$path.count                                  #Examine path counts
pc$paths.bydyad                                #Examine dyadic paths

#Obtain aggregate cycle counts, with co-membership by length
cc<-kcycle.census(g,maxlen=5,tabulate.by.vertex=FALSE,
    cycle.comembership="bylength")
cc$cycle.count                              #Examine cycle counts
cc$cycle.comemb[1,,]                        #Co-membership for 2-cycles
cc$cycle.comemb[2,,]                        #Co-membership for 3-cycles
cc$cycle.comemb[3,,]                        #Co-membership for 4-cycles
```

---

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 graph.
d2	another single graph.
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).
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 exchangeabiliy 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 $|V(G)|^2$*`freeze.time` if `full.neighborhood==TRUE`, 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 `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 `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 *is not recommended for use on substantive problems.* `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 `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., `gscov`, `structdist`) which depend on `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 <buttsc@uci.edu>

References

Butts, C.T. and Carley, K.M. (2005). "Some Simple Algorithms for Structural Comparison." *Computational and Mathematical Organization Theory,* 11(4), 291-305.

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS Working Paper, Carnegie Mellon University.

See Also

gscov, gscor, structdist, sdmat

Examples

```
#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

```
lnam(y, x = NULL, W1 = NULL, W2 = NULL, theta.seed = NULL,
    null.model = c("meanstd", "mean", "std", "none"), method = "BFGS",
    control = list(), tol=1e-10)
```

Arguments

y	a vector of responses.
x	a vector or matrix of covariates; if the latter, each column should contain a single covariate.
W1	one or more (possibly valued) graphs on the elements of y.
W2	one or more (possibly valued) graphs on the elements of y.
theta.seed	an optional seed value for the parameter vector estimation process.

`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`.
`tol`	convergence tolerance for the MLE (expressed as change in deviance).

Details

`lnam` fits the linear network autocorrelation model given by

$$y = W_1 y + X\beta + e, \quad e = W_2 e + \nu$$

where y is a vector of responses, X is a covariate matrix, $\nu \sim N(0, \sigma^2)$,

$$W_1 = \sum_{i=1}^{p} \rho_{1i} W_{1i}, \quad W_2 = \sum_{i=1}^{q} \rho_{2i} W_{2i},$$

and W_{1i}, W_{2i} are (possibly valued) adjacency matrices.

Intuitively, ρ_1 is a vector of "AR"-like parameters (parameterizing the autoregression of each y value on its neighbors in the graphs of W_1) while ρ_2 is a vector of "MA"-like parameters (parameterizing the autocorrelation of each *disturbance* in y on its neighbors in the graphs of W_2). 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 array.
`W2`	if supplied, the W2 array.

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 rho1.
rho1.se	if applicable, the asymptotic standard error for rho1.
rho2	if applicable, the MLE for rho2.
rho2.se	if applicable, the asymptotic standard error for rho2.
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

Leenders, T.Th.A.J. (2002) “Modeling Social Influence Through Network Autocorrelation: Constructing the Weight Matrix” *Social Networks*, 24(1), 21-47.

Anselin, L. (1988) *Spatial Econometrics: Methods and Models.* Norwell, MA: Kluwer.

See Also

lm, optim

Examples

```
## Not run:
#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:
nu<-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)

## End(Not run)
```

---

loadcent *Compute the Load Centrality Scores of Network Positions*

Description

loadcent takes one or more graphs (dat) and returns the load centralities of positions (selected by nodes) within the graphs indicated by g. Depending on the specified mode, load 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

```
loadcent(dat, g = 1, nodes = NULL, gmode = "digraph", diag = FALSE,
    tmaxdev = FALSE, cmode = "directed", geodist.precomp = NULL,
    rescale = FALSE, ignore.eval = TRUE)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g=1.

`nodes`	vector 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`	logical; should self-ties be treated as valid data. Set this true if and only if the data can contain loops. `diag` is `FALSE` by default.
`tmaxdev`	logical; return the theoretical maximum absolute deviation from the maximum nodal centrality (instead of the observed centrality scores)? By default, `tmaxdev==FALSE`.
`cmode`	string indicating the type of load centrality being computed (directed or undirected).
`geodist.precomp`	a `geodist` object precomputed for the graph to be analyzed (optional).
`rescale`	logical; if true, centrality scores are rescaled such that they sum to 1.
`ignore.eval`	logical; ignore edge values when computing shortest paths?

Details

Goh et al.'s *load centrality* (as reformulated by Brandes (2008)) is a betweenness-like measure defined through a hypothetical flow process. Specifically, it is assumed that each vertex sends a unit of some commodity to each other vertex to which it is connected (without edge or vertex capacity constraints), with routing based on a priority system: given an input of flow x arriving at vertex v with destination v', v divides x equally among all neigbors of minumum geodesic distance to the target. The total flow passing through a given v via this process is defined as v's *load*. Load is a potential alternative to betweenness for the analysis of flow structures operating well below their capacity constraints.

Value

A vector of centrality scores.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Brandes, U. (2008). “On Variants of Shortest-Path Betweenness Centrality and their Generic Computation.” *Social Networks*, 30, 136-145.

Goh, K.-I.; Kahng, B.; and Kim, D. (2001). “Universal Behavior of Load Distribution in Scale-free Networks.” *Physical Review Letters*, 87(27), 1-4.

See Also

`betweenness`

Examples

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

`lower.tri.remove` *Remove the Lower Triangles of Adjacency Matrices in a Graph Stack*

Description

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

Usage

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

Arguments

`dat`	one or more input graphs.
`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 set.

Author(s)

Carter T. Butts <buttsc@uci.edu>

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)
```

lubness *Compute Graph LUBness Scores*

Description

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

Usage

```
lubness(dat, g=NULL)
```

Arguments

dat	one or more input graphs.
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 belong to G. An upper bound ℓ 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 ℓ 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

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

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.

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

Krackhardt, David. (1994). "Graph Theoretical Dimensions of Informal Organizations." In K. M. Carley and M. J. Prietula (Eds.), *Computational Organization Theory*, 89-111. Hillsdale, NJ: Lawrence Erlbaum and Associates.

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 collection of input graphs.
mode	one of "row," "col," or "rowcol".
tol	tolerance parameter for the row-column normalization algorithm.
maxiter	maximum iterations for the rwo-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).

Details

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).

Value

The stochasticized adjacency matrices

Warning

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.

Author(s)

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")
```

---

maxflow *Calculate Maximum Flows Between Vertices*

Description

`maxflow` calculates a matrix of maximum pairwise flows within a (possibly valued) input network.

Usage

```
maxflow(dat, src = NULL, sink = NULL, ignore.eval = FALSE)
```

Arguments

`dat`	one or more input graphs.
`src`	optionally, a vector of source vertices; by default, all vertices are selected.
`sink`	optionally, a vector of sink (or target) vertices; by default, all vertices are selected.
`ignore.eval`	logical; ignore edge values (i.e., assume unit capacities) when computing flow?

Details

`maxflow` computes the maximum flow from each source vertex to each sink vertex, assuming infinite vertex capacities and limited edge capacities. If `ignore.eval==FALSE`, supplied edge values are assumed to contain capacity information; otherwise, all non-zero edges are assumed to have unit capacity.

Note that all flows computed here are pairwise – i.e., when computing the flow from v to v', we ignore any other flows which could also be taking place within the network. As a result, it should not be assumed that these flows can be realized *simultaneously*. (For the latter purpose, the values returned by `maxflow` can be treated as upper bounds.)

Value

A matrix of pairwise maximum flows (if multiple sources/sinks selected), or a single maximum flow value (otherwise).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Edmonds, J. and Karp, R.M. (1972). "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." *Journal of the ACM,* 19(2), 248-264.

See Also

`flowbet`, `geodist`

Examples

```
g<-rgraph(10,tp=2/9)                    #Generate a sparse random graph
maxflow(g)                              #Compute all-pairs max flow
```

mutuality *Find the Mutuality of a Graph*

Description

Returns the mutuality scores of the graphs indicated by `g` in `dat`.

Usage

```
mutuality(dat, g=NULL)
```

Arguments

`dat`	one or more input graphs.
`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

Moreno, J.L., and Jennings, H.H. (1938). "Statistics of Social Configurations." *Sociometry*, 1, 342-374.

See Also

`grecip`

Examples

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

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

---

nacf *Sample Network Covariance and Correlation Functions*

Description

nacf computes the sample network covariance/correlation function for a specified variable on a given input network. Moran's I and Geary's C statistics at multiple orders may be computed as well.

Usage

```
nacf(net, y, lag.max = NULL, type = c("correlation", "covariance",
    "moran", "geary"), neighborhood.type = c("in", "out", "total"),
    partial.neighborhood = TRUE, mode = "digraph", diag = FALSE,
    thresh = 0, demean = TRUE)
```

Arguments

net	one or more graphs.
y	a numerical vector, of length equal to the order of net.
lag.max	optionally, the maximum geodesic lag at which to compute dependence (defaults to order net-1).
type	the type of dependence statistic to be computed.
neighborhood.type	the type of neighborhood to be employed when assessing dependence (as per neighborhood).
partial.neighborhood	logical; should partial (rather than cumulative) neighborhoods be employed at higher orders?
mode	"digraph" for directed graphs, or "graph" if net is undirected.
diag	logical; does the diagonal of net contain valid data?
thresh	threshold at which to dichotomize net.
demean	logical; demean y prior to analysis?

Details

`nacf` computes dependence statistics for the vector `y` on network `net`, for neighborhoods of various orders. Specifically, let $\mathbf{A}_i$ be the ith order adjacency matrix of `net`. The sample network autocovariance of $\mathbf{y}$ on $\mathbf{A}_i$ is then given by

$$\sigma_i = \frac{\mathbf{y}^T \mathbf{A}_i \mathbf{y}}{E},$$

where $E = \sum_{(j,k)} A_{ijk}$. Similarly, the sample network autocorrelation in the above case is $\rho_i = \sigma_i/\sigma_0$, where σ_0 is the variance of y. Moran's I and Geary's C statistics are defined in the usual fashion as

$$I_i = \frac{N \sum_{j=1}^{N} \sum_{k=1}^{N} (y_j - \bar{y})(y_k - \bar{y}) A_{ijk}}{E \sum_{j=1}^{N} y_j^2},$$

and

$$C_i = \frac{(N-1) \sum_{j=1}^{N} \sum_{k=1}^{N} (y_j - y_k)^2 A_{ijk}}{2E \sum_{j=1}^{N} (y - \bar{y})^2}$$

respectively, where N is the order of $\mathbf{A}_i$ and $\bar{y}$ is the mean of $\mathbf{y}$.

The adjacency matrix associated with the ith order neighborhood is defined as the identity matrix for order 0, and otherwise depends on the type of neighborhood involved. For input graph $G = (V, E)$, let the *base relation*, R, be given by the underlying graph of G (i.e., $G \cup G^T$) if total neighborhoods are sought, the transpose of G if incoming neighborhoods are sought, or G otherwise. The partial neighborhood structure of order $i > 0$ on R is then defined to be the digraph on V whose edge set consists of the ordered pairs (j, k) having geodesic distance i in R. The corresponding cumulative neighborhood is formed by the ordered pairs having geodesic distance less than or equal to i in R. For purposes of `nacf`, these neighborhoods are calculated using `neighborhood`, with the specified parameters (including dichotomization at `thresh`).

The return value for `nacf` is the selected dependence statistic, calculated for each neighborhood structure from order 0 (the identity) through order `lag.max` (or $N-1$, if `lag.max==NULL`). This vector can be used much like the conventional autocorrelation function, to identify dependencies at various lags. This may, in turn, suggest a starting point for modeling via routines such as `lnam`.

Value

A vector containing the dependence statistics (ascending from order 0).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Geary, R.C. (1954). "The Contiguity Ratio and Statistical Mapping." *The Incorporated Statistician,* 5: 115-145.

Moran, P.A.P. (1950). "Notes on Continuous Stochastic Phenomena." *Biometrika,* 37: 17-23.

See Also

`geodist`, `gapply`, `neighborhood`, `lnam`, `acf`

Examples

```
#Create a random graph, and an autocorrelated variable
g<-rgraph(50,tp=4/49)
y<-qr.solve(diag(50)-0.8*g,rnorm(50,0,0.05))

#Examine the network autocorrelation function
nacf(g,y)                          #Partial neighborhoods
nacf(g,y,partial.neighborhood=FALSE)  #Cumulative neighborhoods

#Repeat, using Moran's I on the underlying graph
nacf(g,y,type="moran")
nacf(g,y,partial.neighborhood=FALSE,type="moran")
```

`neighborhood` *Compute Neighborhood Structures of Specified Order*

Description

For a given graph, returns the specified neighborhood structure at the selected order(s).

Usage

```
neighborhood(dat, order, neighborhood.type = c("in", "out", "total"),
    mode = "digraph", diag = FALSE, thresh = 0, return.all = FALSE,
    partial = TRUE)
```

Arguments

`dat`	one or more graphs.
`order`	order of the neighborhood to extract.
`neighborhood.type`	neighborhood type to employ.
`mode`	`"digraph"` if `dat` is directed, otherwise `"graph"`.
`diag`	logical; do the diagonal entries of `dat` contain valid data?
`thresh`	dichotomization threshold to use for `dat`; edges whose values are greater than `thresh` are treated as present.
`return.all`	logical; return neighborhoods for all orders up to `order`?
`partial`	logical; return partial (rather than cumulative) neighborhoods?

Details

The adjacency matrix associated with the ith order neighborhood is defined as the identity matrix for order 0, and otherwise depends on the type of neighborhood involved. For input graph $G = (V, E)$, let the *base relation*, R, be given by the underlying graph of G (i.e., $G \cup G^T$) if total neighborhoods are sought, the transpose of G if incoming neighborhoods are sought, or G otherwise. The partial neighborhood structure of order $i > 0$ on R is then defined to be the digraph on V whose edge set consists of the ordered pairs (j, k) having geodesic distance i in R. The corresponding cumulative neighborhood is formed by the ordered pairs having geodesic distance less than or equal to i in R.

Neighborhood structures are commonly used to parameterize various types of network autocorrelation models. They may also be used in the calculation of certain types of local structural indices; `gapply` provides an alternative function which can be used for this purpose.

Value

An array or adjacency matrix containing the neighborhood structures (if `dat` is a single graph); if `dat` contains multiple graphs, then a list of such structures is returned.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`gapply`, `nacf`

Examples

```
#Draw a random graph
g<-rgraph(10,tp=2/9)

#Show the total partial out-neighborhoods
neigh<-neighborhood(g,9,neighborhood.type="out",return.all=TRUE)
par(mfrow=c(3,3))
for(i in 1:9)
  gplot(neigh[i,,],main=paste("Partial Neighborhood of Order",i))

#Show the total cumulative out-neighborhoods
neigh<-neighborhood(g,9,neighborhood.type="out",return.all=TRUE,
    partial=FALSE)
par(mfrow=c(3,3))
for(i in 1:9)
  gplot(neigh[i,,],main=paste("Cumulative Neighborhood of Order",i))
```

netcancor — *Canonical Correlation for Labeled Graphs*

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

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

Arguments

`y`	one or more input graphs.
`x`	one or more input graphs.
`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 cancor procedure with more interpretable output; the new version will handle arbitrary labeling as well.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

See Also

gcor, cugtest, qaptest, cancor

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
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, intercept=TRUE, mode="digraph", diag=FALSE,
    nullhyp=c("qap", "qapspp", "qapy", "qapx", "qapallx",
    "cugtie", "cugden", "cuguman", "classical"),
    test.statistic = c("t-value", "beta"), tol=1e-7,
    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	stack of independent network variables. Note that NAs are permitted, as is dichotomous data.
intercept	logical; should an intercept term be added?
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	logical; should the diagonal 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.
test.statistic	string indicating the test statistic to be used for the Monte Carlo procedures.
tol	tolerance parameter for qr.solve.
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

netlm performs an OLS linear network regression of the graph y on the graphs in x. 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:

$$\mathbf{A_y} = b_0\mathbf{A_1} + b_1\mathbf{A_{x_1}} + b_2\mathbf{A_{x_2}} + \ldots + \mathbf{Z}$$

where $\mathbf{A_y}$ is the dependent adjacency matrix, $\mathbf{A_{x_i}}$ is the ith independent adjacency matrix, $\mathbf{A_1}$ is an n x n matrix of 1's, and $\mathbf{Z}$ is an n x n matrix of independent normal random variables with mean 0 and variance σ^2. Clearly, this model is nonoptimal when $\mathbf{A_y}$ 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.)

Because of the frequent presence of row/column/block autocorrelation in network data, classical hull hypothesis tests (and associated standard errors) are generally suspect. Further, it is sometimes of interest to compare fitted parameter values to those arising from various baseline models (e.g., uniform random graphs conditional on certain observed statistics). The tests supported by netlm are as follows:

classical tests based on classical asymptotics.

cug conditional uniform graph test (see cugtest) controlling for order.

cugden conditional uniform graph test, controlling for order and density.

cugtie conditional uniform graph test, controlling for order and tie distribution.

qap QAP permutation test (see qaptest); currently identical to qapspp.

qapallx QAP permutation test, using independent x-permutations.

qapspp QAP permutation test, using Dekker's "semi-partialling plus" procedure.

qapx QAP permutation test, using (single) x-permutations.

qapy QAP permutation test, using y-permutations.

The statistic to be employed in the above tests may be selected via test.statistic. By default, the t-statistic (rather than estimated coefficient) is used, as this is more approximately pivotal;

coefficient-based tests are not recommended for QAP null hypotheses, although they are provided here for legacy purposes.

Note that interpretation of quantiles for single coefficients can be complex in the presence of multicollinearity or third variable effects. qapspp is generally recommended for most multivariable analyses, as it is known to be fairly robust to these conditions. 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

Dekker, D.; Krackhardt, D.; Snijders, T.A.B. (2007). “Sensitivity of MRQAP Tests to Collinearity and Autocorrelation Conditions.” *Psychometrika*, 72(4), 563-581.

Dekker, D.; Krackhardt, D.; Snijders, T.A.B. (2003). “Mulicollinearity Robust QAP for Multiple Regression.” CASOS Working Paper, Carnegie Mellon University.

Krackhardt, D. (1987). “QAP Partialling as a Test of Spuriousness.” *Social Networks*, 9 171-186.

Krackhardt, D. (1988). “Predicting With Networks: Nonparametric Multiple Regression Analyses of Dyadic Data.” *Social Networks*, 10, 359-382.

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 set x. The resulting fits (and coefficients) are then tested against the indicated null hypothesis.

Usage

```
netlogit(y, x, intercept=TRUE, mode="digraph", diag=FALSE,
    nullhyp=c("qap", "qapspp", "qapy", "qapx", "qapallx",
    "cugtie", "cugden", "cuguman", "classical"), test.statistic =
    c("z-value","beta"), tol=1e-7, reps=1000)
```

Arguments

y	dependent network variable. NAs are allowed, and the data should be dichotomous.
x	the stack of independent network variables. Note that NAs are permitted, as is dichotomous data.
intercept	logical; should an intercept term be fitted?
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.
test.statistic	string indicating the test statistic to be used for the Monte Carlo procedures.
tol	tolerance parameter for qr.solve.
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.fit routine. netlogit handles vectorization, sets up glm options, and deals with null hypothesis testing; the actual fitting is taken care of by glm.fit.

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.

Because of the frequent presence of row/column/block autocorrelation in network data, classical hull hypothesis tests (and associated standard errors) are generally suspect. Further, it is sometimes of interest to compare fitted parameter values to those arising from various baseline models (e.g., uniform random graphs conditional on certain observed statistics). The tests supported by `netlogit` are as follows:

`classical` tests based on classical asymptotics.

`cug` conditional uniform graph test (see `cugtest`) controlling for order.

`cugden` conditional uniform graph test, controlling for order and density.

`cugtie` conditional uniform graph test, controlling for order and tie distribution.

`qap` QAP permutation test (see `qaptest`); currently identical to `qapspp`.

`qapallx` QAP permutation test, using independent x-permutations.

`qapspp` QAP permutation test, using Dekker's "semi-partialling plus" procedure.

`qapx` QAP permutation test, using (single) x-permutations.

`qapy` QAP permutation test, using y-permutations.

Note that interpretation of quantiles for single coefficients can be complex in the presence of multicollinearity or third variable effects. Although `qapspp` is known to be robust to these conditions in the OLS case, there are no equivalent results for logistic regression. Caution is thus advised.

The statistic to be employed in the above tests may be selected via `test.statistic`. By default, the z-statistic (rather than estimated coefficient) is used, as this is more approximately pivotal; coefficient-based tests are not recommended for QAP null hypotheses, although they are provided here for legacy purposes.

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

Butts, C.T., and Carley, K.M. (2001). "Multivariate Methods for Interstructural Analysis." CASOS working paper, Carnegie Mellon University.

See Also

`glm`, `netlm`

Examples

```
## Not run:
#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)

## End(Not run)
```

---

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

```
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

Gelman, A.; Carlin, J.B.; Stern, H.S.; and Rubin, D.B. (1995). *Bayesian Data Analysis.* London: Chapman and Hall.

See Also

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(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 — *Find the Number of Possible Ties in a Given Graph or Graph 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 set 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 ith. 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 `permnumth` 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.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`rperm`, `rmperm`

Examples

```
#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]
```

`plot.bbnam` *Plotting for bbnam Objects*

Description

Generates various plots of posterior draws from the `bbnam` model.

Usage

```
## S3 method for class 'bbnam'
plot(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

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T. (1999). "Informant (In)Accuracy and Network Estimation: A Bayesian Approach." CASOS Working Paper, Carnegie Mellon University.

See Also

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(dat,model="actor",nprior=pnet,emprior=pem,epprior=pep,
    burntime=100,draws=100)
#Print a summary of the posterior draws
summary(b)
#Plot the result
plot(b)
```

`plot.blockmodel` *Plotting for blockmodel Objects*

Description

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

Usage

```
## S3 method for class 'blockmodel'
plot(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 <`buttsc@uci.edu`>

References

White, H.C.; Boorman, S.A.; and Breiger, R.L. (1976). "Social Structure from Multiple Networks I: Blockmodels of Roles and Positions." *American Journal of Sociology*, 81, 730-779.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

`blockmodel`, `plot.sociomatrix`

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
```

`plot.cugtest` *Plotting for cugtest Objects*

Description

Plots the distribution of a CUG test statistic.

Usage

```
## S3 method for class 'cugtest'
plot(x, mode="density", ...)
```

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 <buttsc@uci.edu>

References

Anderson, B.S.; Butts, C.T.; and Carley, K.M. (1999). "The Interaction of Size and Density with 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

```
## S3 method for class 'equiv.clust'
plot(x, labels=NULL, ...)
```

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 <buttsc@uci.edu>

References

Breiger, R.L.; Boorman, S.A.; and Arabie, P. (1975). "An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling." *Journal of Mathematical Psychology*, 12, 328-383.

Burt, R.S. (1976). "Positions in Networks." *Social Forces*, 55, 93-122.

Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

equiv.clust, plot.hclust

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)
```

---

plot.lnam *Plotting for lnam Objects*

Description

Generates various diagnostic plots for lnam objects.

Usage

```
## S3 method for class 'lnam'
plot(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

```
## S3 method for class 'qaptest'
plot(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.

Value

None

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Hubert, L.J., and Arabie, P. (1989). "Combinatorial Data Analysis: Confirmatory Comparisons Between Sets of Matrices." *Applied Stochastic Models and Data Analysis*, 5, 273-325.

Krackhardt, D. (1987). "QAP Partialling as a Test of Spuriousness." *Social Networks*, 9 171-186.

Krackhardt, D. (1988). "Predicting With Networks: Nonparametric Multiple Regression Analyses of Dyadic Data." *Social Networks*, 10, 359-382.

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

```
## S3 method for class 'sociomatrix'
plot(x, labels=NULL, drawlab=TRUE, diaglab=TRUE,
    drawlines=TRUE, xlab=NULL, ylab=NULL, cex.lab=1, font.lab=1, col.lab=1,
    scale.values=TRUE, cell.col=gray, ...)

sociomatrixplot(x, labels=NULL, drawlab=TRUE, diaglab=TRUE,
    drawlines=TRUE, xlab=NULL, ylab=NULL, cex.lab=1, font.lab=1, col.lab=1,
    scale.values=TRUE, cell.col=gray, ...)
```

Arguments

`x`	an input graph.
`labels`	a list containing the vectors of row and column labels (respectively); defaults to the row/column labels of `x` (if specified), or otherwise sequential numerical labels.
`drawlab`	logical; add row/column labels to the plot?
`diaglab`	logical; label the diagonal?
`drawlines`	logical; draw lines to mark cell boundaries?

xlab	x axis label.
ylab	y axis label.
cex.lab	optional expansion factor for labels.
font.lab	optional font specification for labels.
col.lab	optional color specification for labels.
scale.values	logical; should cell values be affinely scaled to the [0,1] interval? (Defaults to TRUE.)
cell.col	function taking a vector of cell values as an argument and returning a corresponding vector of colors; defaults to gray.
...	additional arguments to plot.

Details

plot.sociomatrix is particularly valuable for examining large adjacency matrices, whose structure can be non-obvious otherwise. sociomatrixplot is an alias to plot.sociomatrix, and may eventually supersede it.

The cell.col argument can be any function that takes input cell values and returns legal colors; while gray will produce an error for cell values outside the [0,1] interval, user-specified functions can be employed to get other effects (see examples below). Note that, by default, all input cell values are affinely scaled to the [0,1] interval before colors are computed, so scale.values must be set to FALSE to allow access to the raw inputs.

Value

None

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

plot.blockmodel

Examples

```
#Plot a small adjacency matrix
plot.sociomatrix(rgraph(5))

#Plot a much larger one
plot.sociomatrix(rgraph(100), drawlab=FALSE, diaglab=FALSE)

#Example involving a signed, valued graph and custom colors
mycolfun <- function(z){   #Custom color function
    ifelse(z<0, rgb(1,0,0,alpha=1-1/(1-z)), ifelse(z>0,
        rgb(0,0,1,alpha=1-1/(1+z)), rgb(0,0,0,alpha=0)))
}
sg <- rgraph(25) * matrix(rnorm(25^2),25,25)
plot.sociomatrix(sg, scale.values=FALSE, cell.col=mycolfun)  #Blue pos/red neg
```

`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 ($\sqrt{\hat{R}}$) 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 $\sqrt{\hat{R}}$ 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 <buttsc@uci.edu>

References

Gelman, A.; Carlin, J.B.; Stern, H.S.; and Rubin, D.B. (1995). *Bayesian Data Analysis.* London: Chapman and Hall.

Gelman, A., and Rubin, D.B. (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science,* 7, 457-511.

See Also

bbnam

---

prestige *Calculate the Vertex Prestige Scores*

Description

prestige takes one or more graphs (dat) and returns the prestige scores of positions (selected by nodes) within the graphs indicated by g. 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).

Usage

```
prestige(dat, g=1, nodes=NULL, gmode="digraph", diag=FALSE,
    cmode="indegree", tmaxdev=FALSE, rescale=FALSE, tol=1e-07)
```

Arguments

dat	one or more input graphs.
g	integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). By default, g==1.
nodes	vector 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, matrix, or list containing the prestige scores (depending on the number and size of the input graphs).

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 <buttsc@uci.edu>

References

Lin, N. (1976). *Foundations of Social Research*. New York: McGraw Hill.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

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

```
## S3 method for class 'bayes.factor'
print(x, ...)
```

Arguments

`x`	An object of class `bayes.factor`
`...`	Further arguments passed to or from other methods

Value

None

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`bbnam.bf`

`print.bbnam` *Printing for bbnam Objects*

Description

Prints a quick summary of posterior draws from `bbnam`.

Usage

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

Arguments

`x`	A `bbnam` object
`...`	Further arguments passed to or from other methods

Value

None

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

bbnam

---

print.blockmodel *Printing for blockmodel Objects*

Description

Prints a quick summary of a blockmodel object.

Usage

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

Arguments

x	An object of class blockmodel
...	Further arguments passed to or from other methods

Value

None

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

blockmodel

print.cugtest *Printing for cugtest Objects*

Description

Prints a quick summary of objects produced by `cugtest`.

Usage

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

Arguments

`x`	An object of class `cugtest`
`...`	Further arguments passed to or from other methods

Value

None.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`cugtest`

print.lnam *Printing for lnam Objects*

Description

Prints an objsect of class `lnam`

Usage

```
## S3 method for class 'lnam'
print(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 <buttsc@uci.edu>

See Also

lnam

---

print.netcancor *Printing for netcancor Objects*

Description

Prints a quick summary of objects produced by netcancor.

Usage

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

Arguments

x	An object of class netcancor
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

netcancor

print.netlm *Printing for netlm Objects*

Description

Prints a quick summary of objects produced by `netlm`.

Usage

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

Arguments

`x`	An object of class `netlm`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`netlm`

print.netlogit *Printing for netlogit Objects*

Description

Prints a quick summary of objects produced by `netlogit`.

Usage

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

Arguments

`x`	An object of class `netlogit`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

netlogit

---

print.qaptest *Printing for qaptest Objects*

Description

Prints a quick summary of objects produced by qaptest.

Usage

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

Arguments

x	An object of class qaptest
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

qaptest

---

print.summary.bayes.factor *Printing for summary.bayes.factor Objects*

Description

Prints an object of class summary.bayes.factor.

Usage

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

Arguments

x	An object of class summary.bayes.factor
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

summary.bayes.factor

---

print.summary.bbnam *Printing for summary.bbnam Objects*

Description

Prints an object of class summary.bbnam.

Usage

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

Arguments

x	An object of class summary.bbnam
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

bbnam

---

print.summary.blockmodel
Printing for summary.blockmodel Objects

Description

Prints an object of class summary.blockmodel.

Usage

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

Arguments

`x`	An object of class `summary.blockmodel`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`summary.blockmodel`

`print.summary.cugtest` *Printing for summary.cugtest Objects*

Description

Prints an object of class `summary.cugtest`.

Usage

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

Arguments

`x`	An object of class `summary.cugtest`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`summary.cugtest`

print.summary.lnam *Printing for summary.lnam Objects*

Description

Prints an object of class `summary.lnam`.

Usage

```
## S3 method for class 'summary.lnam'
print(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

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

Arguments

`x`	An object of class `summary.netcancor`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`summary.netcancor`

`print.summary.netlm` *Printing for summary.netlm Objects*

Description

Prints an object of class `summary.netlm`.

Usage

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

Arguments

`x`	An object of class `summary.netlm`
`...`	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`summary.netlm`

print.summary.netlogit *Printing for summary.netlogit Objects*

Description

Prints an object of class `summary.netlogit`.

Usage

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

Arguments

x	An object of class `summary.netlogit~`
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`summary.netlogit`

print.summary.qaptest *Printing for summary.qaptest Objects*

Description

Prints an object of class `summary.qaptest`.

Usage

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

Arguments

x	An object of class `summary.qaptest`
...	Further arguments passed to or from other methods

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

summary.qaptest

---

pstar *Fit a p*/ERG Model Using a Logistic Approximation*

Description

Fits a p*/ERG model to the graph in `dat` containing the effects listed in `effects`. The result is returned as a `glm` object.

Usage

```
pstar(dat, effects=c("choice", "mutuality", "density", "reciprocity",
    "stransitivity", "wtransitivity", "stranstri",  "wtranstri",
    "outdegree", "indegree", "betweenness", "closeness",
    "degcentralization", "betcentralization", "clocentralization",
    "connectedness", "hierarchy", "lubness", "efficiency"),
    attr=NULL, memb=NULL, diag=FALSE, mode="digraph")
```

Arguments

`dat`	a single graph
`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* MLE 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 pseudolikelihood 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 <buttsc@uci.edu>

References

Anderson, C.; Wasserman, S.; and Crouch, B. (1999). "A p* Primer: Logit Models for Social Networks. *Social Networks,* 21,37-66.

Holland, P.W., and Leinhardt, S. (1981). "An Exponential Family of Probability Distributions for Directed Graphs." *Journal of the American statistical Association*, 81, 51-67.

Wasserman, S., and Pattison, P. (1996). "Logit Models and Logistic Regressions for Social Networks: I. An introduction to Markov Graphs and p*." *Psychometrika,* 60, 401-426.

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 -- use ONLY as heuristics!!!
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	graphs to be analyzed. Though one could in principle use a single 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 qaptest procedure returns a 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>=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.

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 underling structures, QAP will tend to given non-significant results. (In fact, it is *impossible* to obtain a one-tailed significance level in excess of $\max_{g \in \{G,H\}} \frac{|Aut(g)|}{|Perm(g)|}$ 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 `qaptest`, containing

`testval`	The observed value of the test statistic.
`dist`	A vector containing the Monte Carlo draws.
`pgreq`	The proportion of draws which were greater than or equal to the observed value.
`pleeq`	The proportion of draws which were less than or equal to the observed value.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Anderson, B.S.; Butts, C.T.; and Carley, K.M. (1999). "The Interaction of Size and Density with Graph-Level Indices." *Social Networks*, 21(3), 239-267.

Hubert, L.J., and Arabie, P. (1989). "Combinatorial Data Analysis: Confirmatory Comparisons Between Sets of Matrices." *Applied Stochastic Models and Data Analysis*, 5, 273-325.

Krackhardt, D. (1987). "QAP Partialling as a Test of Spuriousness." *Social Networks*, 9 171-186.

Krackhardt, D. (1988). "Predicting With Networks: Nonparametric Multiple Regression Analyses of Dyadic Data." *Social Networks*, 10, 359-382.

See Also

`cugtest`

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)
```

`reachability` *Find the Reachability Matrix of a Graph*

Description

`reachability` takes one or more (possibly directed) graphs as input, producing the associated reachability matrices.

Usage

```
reachability(dat, geodist.precomp=NULL, return.as.edgelist=FALSE, na.omit=TRUE)
```

Arguments

`dat`	one or more graphs (directed or otherwise).
`geodist.precomp`	optionally, a precomputed `geodist` object.
`return.as.edgelist`	logical; return the result as an sna edgelist?
`na.omit`	logical; omit missing edges when computing reach?

Details

For a digraph $G = (V, E)$ with vertices i and j, let P_{ij} represent a directed ij path. Then the (di)graph

$$R = \left(V\left(G\right), \left\{\left(i, j\right) : i, j \in V\left(G\right), P_{ij} \in G\right\}\right)$$

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).

By default, `reachability` will try to build the reachability graph using an internal sparse graph approximation; this is no help on fully connected graphs (but not a lot worse than using an adjacency matrix), but will result in considerable savings for large graphs that are heavily fragmented. (The intended design tradeoff is thus that one pays a small cost on the usually cheap cases, in exchange for much greater efficiency on the cases that would otherwise be prohibitively expensive.) If `geodist.precomp` is given, however, the $O(N^2)$ cost of an adjacency matrix representation has already been paid, and we simply employ what we are given – so, if you want to force the internal use of adjacency matrices, just pass a `geodist` object. Because the internal representation used is otherwise list based, using `return.as.edgelist=TRUE` will save resources; if you are using `reachability` as part of a more complex series of calls, it is thus recommended that you both pass and return sna edgelists unless you have a good reason not to do so.

When set, `na.omit` results in missing edges (i.e., edges with `NA` values) being removed prior to computation. Since paths are not recomputed when `geodist.precomp` is passed, this option is only active when `geodist.precomp==NULL`; if this behavior is desired and precomputed distances are being used, such edges should be removed prior to the `geodist` call.

Value

A reachability matrix, or the equivalent edgelist representation

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

`geodist`

Examples

```
#Find the reachability matrix for a sparse random graph
g<-rgraph(10,tprob=0.15)
rg<-reachability(g)
g  #Compare the two structures
rg

#Compare to the output of geodist
all(rg==(geodist(g)$counts>0))
```

read.dot *Read Graphviz DOT Files*

Description

Reads network information in Graphviz's DOT format, returning an adjacency matrix.

Usage

```
read.dot(...)
```

Arguments

`...`	The name of the file whence to import the data, or else a connection object (suitable for processing by `readLines`.

Details

The Graphviz project's DOT language is a simple but flexible tool for describing graphs. See the included reference for details.

Value

The imported graph, in adjacency matrix form.

Author(s)

Matthijs den Besten <matthijs.denbesten@gmail.com>

References

Graphviz Project. "The DOT Language." http://www.graphviz.org/doc/info/lang.html

See Also

`read.nos`, `write.nos`, `write.dl`

`read.nos` *Read (N)eo-(O)rg(S)tat Input Files*

Description

Reads an input file in NOS format, returning the result as a graph set.

Usage

```
read.nos(file, return.as.edgelist = FALSE)
```

Arguments

`file`	the file to be imported
`return.as.edgelist`	logical; should the resulting graphs be returned in sna edgelist format?

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 set (in adjacency array or edgelist form).

Note

`read.nos` currently ignores the coloring information.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`write.nos`, `scan`, `read.table`

`redist` *Find a Matrix of Distances Between Positions Based on Regular Equivalence*

Description

`redist` uses the graphs indicated by `g` in `dat` to assess the extent to which each vertex is regularly equivalent; `method` determines the measure of approximate equivalence which is used (currently, only CATREGE).

Usage

```
redist(dat, g = NULL, method = c("catrege"), mode = "digraph",
    diag = FALSE, seed.partition = NULL, code.diss = TRUE, ...)
```

Arguments

`dat`	a graph or set thereof.
`g`	a vector indicating which elements of `dat` should be examined (by default, all are used).
`method`	method to use when assessing regular equivalence (currently, only `"catrege"`).
`mode`	`"digraph"` for directed data, otherwise `"graph"`.
`diag`	logical; should diagonal entries (loops) should be treated as meaningful data?
`seed.partition`	optionally, an initial equivalence partition to “seed” the CATREGE algorithm.
`code.diss`	logical; return as dissimilarities (rather than similarities)?
`...`	additional parameters (currently ignored).

Details

`redist` provides a basic tool for assessing the (approximate) regular equivalence of actors. Two vertices i and j are said to be regularly equivalent with respect to role assignment r if $\{r(u) : u \in N^+(i)\} = \{r(u) : u \in N^+(j)\}$ and $\{r(u) : u \in N^-(i)\} = \{r(u) : u \in N^-(j)\}$, where N^+ and N^- denote out- and in-neighborhoods (respectively). RE similarity/difference scores are computed by `method`, currently Borgatti and Everett's CATREGE algorithm (which is based on the multiplex maximal regular equivalence on G and its transpose). The "distance" between positions in this case is the inverse of the number of iterative refinements of the initial equivalence (i.e., role) structure required to allocate the positions to regularly equivalent roles (with 0 indicating positions which ultimately belong in the same role). By default, the initial equivalence structure is one in which all vertices are treated as occupying the same role; the `seed.partition` option can be used to impose alternative constraints. From this initial structure, vertices within the same role having non-identical mixes of neighbor types are re-allocated to different roles (where "neighbor type" is initially due to the pattern of (possibly valued) in- and out-ties, cross-classified by current alter type). This procedure is then iterated until no further division of roles is necessary to satisfy the regularity condition.

Once the similarities/differences are calculated, the results can be used with a clustering routine (such as `equiv.clust`) or an MDS (such as `cmdscale`) to identify the underlying role structure.

Value

A matrix of similarity/difference scores.

Note

The maximal regular equivalence is often very uninteresting (i.e., degenerate) for unvalued, undirected graphs. An exogenous constraint (e.g., via the `seed.partition`) may be required to uncover a more useful refinement of the unconstrained maximal equivalence.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Borgatti, S.P. and Everett, M.G. (1993). "Two Algorithms for Computing Regular Equivalence." *Social Networks*, 15, 361-376.

See Also

`sedist`, `equiv.clust`

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
```

```
#Get RE distances
g.re<-redist(g)

#Plot a metric MDS of vertex positions in two dimensions
plot(cmdscale(as.dist(g.re)))
```

---

rgbn *Draw from a Skvoretz-Fararo Biased Net Process*

Description

Produces a series of draws from a Skvoretz-Fararo biased net process using a (pseudo) Gibbs sampler or exact sampling procedure.

Usage

```
rgbn(n, nv, param = list(pi=0, sigma=0, rho=0, d=0.5, delta=0),
    burn = nv*nv*5*100, thin = nv*nv*5, maxiter = 1e7,
    method = c("mcmc","cftp"), dichotomize.sib.effects = FALSE,
    return.as.edgelist = FALSE)
```

Arguments

`n`	number of draws to take.
`nv`	number of vertices in the graph to be simulated.
`param`	a list containing the biased net parameters (as described below); d may be given as a scalar or as an `nv x nv` matrix of edgewise baseline edge probabilities.
`burn`	for the Gibbs sampler, the number of burn-in draws to take (and discard).
`thin`	the thinning parameter for the Gibbs sampler.
`maxiter`	for the CFTP method, the number of iterations to try before giving up.
`method`	`"mcmc"` for the Gibbs sampler, or `"cftp"` for the exact sampling procedure.
`dichotomize.sib.effects`	logical; should sibling and double role effects be dichotomized?
`return.as.edgelist`	logical; should the simulated draws be returned in edgelist format?

Details

The biased net model stems from early work by Rapoport, who attempted to model networks via a hypothetical "tracing" process. This process may be described loosely as follows. One begins with a small "seed" set of vertices, each member of which is assumed to nominate (generate ties to) other members of the population with some fixed probability. These members, in turn, may nominate new members of the population, as well as members who have already been reached. Such nominations may be "biased" in one fashion or another, leading to a non-uniform growth process.

While the original biased net model depends upon the tracing process, a local interpretation has been put forward by Skvoretz and colleagues in recent years. Using the standard four-parameter process, the conditional probability of an (i, j) edge given all other edges in a random graph G can be approximated as

$$\Pr(i \to j|G_{-ij}) \approx 1 - (1-\rho)^z (1-\sigma)^y (1-\pi)^x (1-d_{ij})$$

where $x = 1$ iff $j \to i$ (and 0 otherwise), y is the number of vertices $k \neq i, j$ such that $k \to i, k \to j$, and $z = 1$ iff $x = 1$ and $y > 0$ (and 0 otherwise). Thus, x is the number of potential *parent bias* events, y is the number of potential *sibling bias* events, and z is the number of potential *double role bias* events. d_{ij} is the probability of the baseline edge event; note that an edge arises if the baseline event or any bias event occurs, and all events are assumed conditionally independent. Written in this way, it is clear that the edges of G are conditionally independent if they share no endpoint. Thus, a model with the above structure should be a subfamily of the Markov graphs.

One potential problem with the above structure is that the hypothetical probabilities implied by the model are not guaranteed to be consistent - that is, the conditions under which there exists a joint pmf with the implied full conditionals are currently unknown (and may be restrictive). The interpretation of the above as exact conditional probabilities is thus potentially problematic. However, a well-defined process can be constructed by interpreting the above as transition probabilities for a Markov chain that evolves by updating a randomly selected edge variable at each time point; this is a Gibbs sampler for the implied joint pmf where it exists, and otherwise an irreducible and aperiodic Markov chain with a well-defined equilibrium distribution.

In the above process, all events act to promote the formation of edges; it is also possible to define events that inhibit them. For instance, consider a *satiation* event that, if it occurs, forbids the creation of an $i \to j$ edge; we assume that a potential satiation event occurs every time i emits an edge to some other vertex. The associated approximate conditional (i.e., transition probability) is

$$\Pr(i \to j|G_{-ij}) \approx (1-\delta)^w \left(1 - (1-\rho)^z (1-\sigma)^y (1-\pi)^x (1-d_{ij})\right)$$

where w is the outdegree of i in G_{-ij} and δ is the probability of the satiation event. The net effect of satiation is to suppress edge formation (in roughly geometric fashion) on high degree nodes. This may be useful in preventing degeneracy when using sigma and rho effects. Degeneracy can also be reduced by employing the `dichotomize.sib.effects` argument, which counts only the first shared partner's contribution towards sibling and double role effects.

It should be noted that the above process is not entirely consistent with the tracing-based model, which is itself not uniformly well-specified in the literature. For this reason, the local model is referred to here as a Skvoretz-Fararo graph process. One significant advantage of this process is that it is well-defined, and easily simulated: the above equation can be used to form the basis of a (pseudo-) Gibbs sampler, which is used by $rgbn$ to take draws from the (local) biased net model. Burn-in and thinning are controlled by the corresponding arguments; since degeneracy is common with models of this type, it is advisable to check for adequate mixing. An alternative simulation strategy is the exact sampling procedure of Butts (2018), which employs a form of coupling from the past (CFTP). The CFTP method generates exact, independent draws from the equilibrium distribution of the biased net process (up to numerical limits), but can be slow to attain coalescence (and does not currently support satiation events). Setting `maxiter` to smaller values limits the search depth employed, at the possible cost of biasing the resulting sample.

Value

An adjacency array containing the simulated graphs.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T. (2018). “A Perfect Sampling Method for Exponential Family Random Graph Models.” *Journal of Mathematical Sociology*, 42(1), 17-36.

Rapoport, A. (1957). “A Contribution to the Theory of Random and Biased Nets.” *Bulletin of Mathematical Biophysics,* 15, 523-533.

Skvoretz, J.; Fararo, T.J.; and Agneessens, F. (2004). “Advances in Biased Net Theory: Definitions, Derivations, and Estimations.” *Social Networks,* 26, 113-139.

See Also

bn

Examples

```
#Generate draws with low density and no biases
g1<-rgbn(50,10,param=list(pi=0, sigma=0, rho=0, d=0.17))
apply(dyad.census(g1),2,mean) #Examine the dyad census

#Add a reciprocity bias
g2<-rgbn(50,10,param=list(pi=0.5, sigma=0, rho=0, d=0.17))
apply(dyad.census(g2),2,mean) #Compare with g1

#Alternately, add a sibling bias
g3<-rgbn(50,10,param=list(pi=0.0, sigma=0.3, rho=0, d=0.17))
mean(gtrans(g3))              #Compare transitivity scores
mean(gtrans(g1))
```

---

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,
    return.as.edgelist = 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	logical; should loops be allowed?
return.as.edgelist	logical; should the resulting graphs be returned in edgelist form?

Details

rgnm returns draws from the density-conditioned uniform random graph first popularized by the famous work of Erdos and Renyi (the $G(N, M)$ process). In particular, the pmf of a $G(N, M)$ process is given by

$$p(G = g|N, M) = \binom{E_m}{M}^{-1}$$

where E_m is the maximum number of edges in the graph. (E_m is equal to `nv*(nv-diag)/(1+(mode=="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 Erdos and Renyi) and the U|MAN model of dyadic independence. These are implemented within sna as rgraph and rgnm, respectively.

Value

A matrix or array containing the drawn adjacency matrices

Note

The famous mathematicians referenced in this man page now have misspelled names, due to R's difficulty with accent marks.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Erdos, P. and Renyi, A. (1960). "On the Evolution of Random Graphs." *Public Mathematical Institute of Hungary Academy of Sciences,* 5:17-61.

See Also

rgraph, rguman

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))
```

---

rgnmix *Draw Mixing-Conditioned Random Graphs*

Description

`rgnmix` generates random draws from a mixing-conditioned uniform random graph distribution.

Usage

```
rgnmix(n, tv, mix, mode = "digraph", diag = FALSE,
    method = c("probability", "exact"), return.as.edgelist = FALSE)
```

Arguments

`n`	the number of graphs to generate.
`tv`	a vector of types or classes (one entry per vertex), corresponding to the rows and columns of `mix`. (Note that the total number of vertices generated will be `length(tv)`.)
`mix`	a class-by-class mixing matrix, containing either mixing rates (for `method=="probability"`) or edge counts (for `method=="exact"`).
`mode`	`"digraph"` for directed graphs, or `"graph"` for undirected graphs.
`diag`	logical; should loops be allowed?
`method`	the generation method to use. `"probability"` results in a Bernoulli edge distribution (conditional on the underlying rates), while `"exact"` results in a uniform draw conditional on the exact per-block edge distribution.
`return.as.edgelist`	logical; should the resulting graphs be returned in sna edgelist form?

Details

The generated graphs (in either adjacency or edgelist form).

Value

`rgnmix` draws from a simple generalization of the Erdos-Renyi N,M family (and the related N,p family), generating graphs with fixed expected or realized mixing rates. Mixing is determined by the `mix` argument, which must contain a class-by-class matrix of mixing rates (either edge probabilities or number of realized edges). The class for each vertex is specified in `tv`, whose entries must correspond to the rows and columns of `mix`. The resulting functionality is much like `blockmodel.expand`, although more general (and in some cases more efficient).

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

rguman, rgnm, blockmodel.expand

Examples

```
#Draw a random mixing matrix
mix<-matrix(runif(9),3,3)

#Generate a graph with 4 members per class
g<-rgnmix(1,rep(1:3,each=4),mix)
plot.sociomatrix(g)                                   #Visualize the result

#Repeat the exercise, using the exact method
mix2<-round(mix*8)                                    #Draw an exact matrix
g<-rgnmix(1,rep(1:3,each=4),mix2,method="exact")
plot.sociomatrix(g)
```

---

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, return.as.edgelist=FALSE)
```

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

`return.as.edgelist`
logical; should the resulting graphs be returned in edgelist form?

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

Note

The famous mathematicians referenced in this man page now have misspelled names, due to R's difficulty with accent marks.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Erdos, P. and Renyi, A. (1960). "On the Evolution of Random Graphs." *Public Mathematical Institute of Hungary Academy of Sciences,* 5:17-61.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

See Also

`rmperm`, `rgnm`, `rguman`

Examples

```
#Generate three graphs with different densities
g<-rgraph(10,3,tprob=c(0.1,0.9,0.5))

#Generate from a matrix of Bernoulli parameters
g.p<-matrix(runif(25,0,1),nrow=5)
g<-rgraph(5,2,tprob=g.p)
```

---

rguman — *Draw Dyad Census-Conditioned Random Graphs*

Description

rguman generates random draws from a dyad census-conditioned uniform random graph distribution.

Usage

```
rguman(n, nv, mut = 0.25, asym = 0.5, null = 0.25,
    method = c("probability", "exact"), return.as.edgelist = FALSE)
```

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.
return.as.edgelist	logical; should the resulting graphs be returned in edgelist form?

Details

A simple generalization of the Erdos-Renyi 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).

Value

A matrix or array containing the drawn adjacency matrices

Note

The famous mathematicians referenced in this man page now have misspelled names, due to R's difficulty with accent marks.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Holland, P.W. and Leinhardt, S. (1976). "Local Structure in Social Networks." In D. Heise (Ed.), *Sociological Methodology*, pp 1-45. San Francisco: Jossey-Bass.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

`rgraph`, `rgnm`, `dyad.census`

Examples

```
#Show some examples of extreme U|MAN 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|MAN 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, return.as.edgelist = FALSE)
rewire.ud(g, p, return.as.edgelist = FALSE)
rewire.ws(g, p, return.as.edgelist = FALSE)
```

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.
return.as.edgelist	logical; should the resulting graphs be returned in edgelist form?

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.

rgws produces independent draws from the above process, returning them as an adjacency matrix (if n==1) or array (otherwise). 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. rewire.ud is similar to 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

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 <buttsc@uci.edu>

References

Watts, D. and Strogatz, S. (1998). "Collective Dynamics of Small-world Networks." *Nature,* 393:440-442.

See Also

rgnm, 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% rewiring
gplot(rgws(1,100,1,2,0.05))  #5% rewiring
gplot(rgws(1,100,1,2,0.1))   #10% rewiring
gplot(rgws(1,100,1,2,1))     #100% rewiring

#Start with a simple graph, then rewire it
g<-matrix(0,50,50)
g[1,]<-1; g[,1]<-1    #Create a star
gplot(g)
gplot(rewire.ws(g,0.05))  #5% rewiring
```

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

```
rmperm(m)
```

Arguments

`m`	a matrix, or stack thereof (or a graph set, for that matter).

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 <buttsc@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 Stack*

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=NULL, ...)
```

Arguments

dat	graph set to be analyzed.
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

$$d_S\left(G, H \left| L_G, L_H\right.\right) = \min_{L_G, L_H} d\left(\ell\left(G\right), \ell\left(H\right)\right)$$

where L_G is the set of accessible permutations/labelings of G, and $\ell(G)$ is a permuation/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 exchangeabiliy 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 `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 matrix of distances (or an object of class `dist`)

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

For most applications, `sdmat` is dominated by `structdist`; the former is retained largely for reasons of compatibility.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Butts, C.T. and Carley, K.M. (2005). “Some Simple Algorithms for Structural Comparison.” *Computational and Mathematical Organization Theory,* 11(4), 291-305.

Butts, C.T., and Carley, K.M. (2001). “Multivariate Methods for Interstructural Analysis.” CASOS Working Paper, Carnegie Mellon University.

See Also

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?
sdmat(g,exchange.list=1:5)

#What are the structural distances between the underlying unlabeled
#graphs?
sdmat(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 set thereof.
g	a vector indicating which elements of dat should be examined.
method	one of "correlation", "euclidean", "hamming", or "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

Breiger, R.L.; Boorman, S.A.; and Arabie, P. (1975). “An Algorithm for Clustering Relational Data with Applications to Social Network Analysis and Comparison with Multidimensional Scaling.” *Journal of Mathematical Psychology*, 12, 328-383.

Burt, R.S. (1976). “Positions in Networks.” *Social Forces*, 55, 93-122.

Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

See Also

equiv.clust, 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

#Get SE distances
```

```
g.se<-sedist(g)

#Plot a metric MDS of vertex positions in two dimensions
plot(cmdscale(as.dist(g.se)))
```

---

sna *Tools for Social Network Analysis*

Description

sna is a package containing a range of tools for social network analysis. Supported functionality includes node and graph-level indices, structural distance and covariance methods, structural equivalence detection, p* modeling, random graph generation, and 2D/3D network visualization (among other things).

Details

Network data for sna routines can (except as noted otherwise) appear in any of the following forms:

- adjacency matrices (dimension N x N);
- arrays of adjacency matrices, aka "graph stacks" (dimension m x N x N);
- sna edge lists (see below);
- sparse matrix objects (from the SparseM package);
- network objects (from the network package); or
- lists of adjacency matrices/arrays, sparse matrices, and/or network objects.

Within the package documentation, the term "graph" is used generically to refer to any or all of the above (with multiple graphs being referred to as a "graph stack"). Note that usage of sparse matrix objects requires that the SparseM package be installed. (No additional packages are required for use of adjacency matrices/arrays or lists thereof, though the network package, on which sna depends as of 2.4, is used for network objects.) In general, sna routines attempt to make intelligent decisions regarding the processing of multiple graphs, but common sense is always advised; certain functions, in particular, have more specific data requirements. Calling sna functions with inappropriate input data can produce "interesting" results.

One special data type supported by the sna package (as of version 2.0) is the *sna edgelist*. This is a simple data format that is well-suited to representing large, sparse graphs. (As of version 2.0, many - now most - package routines also process data in this form natively, so using it can produce significant savings of time and/or memory. Prior to 2.0, all package functions coerced input data to adjacency matrix form.) An sna edgelist is a three-column matrix, containing (respectively) senders, receivers, and values for each edge in the graph. (Unvalued edges should have a value of 1.) Note that this form is invariant to the number of edges in the graph: if there are no edges, then the edgelist is a degenerate matrix of dimension 0 by 3. Edgelists for undirected graphs should be coded as fully mutual digraphs (as would be the case with an adjacency matrix), with two edges per dyad (one (i,j) edge, and one (j,i) edge). Graph size for an sna edgelist matrix is indicated by a mandatory numeric attribute, named "n". Vertex names may be optionally specified by a vector-valued attribute named "vnames". In the case of two-mode data (i.e., data with an enforced bipartition), it is possible to indicate this status via the optional "bipartite" attribute. Vertices in a two-mode edgelist should be grouped in mode order, with "n" equal to the total number of vertices (across both modes) and "bipartite" equal to the number of vertices in the first mode.

Direct creation of sna edgelists can be performed by creating a three-column matrix and using the attr function to create the required "n" attribute. Alternately, the function as.edgelist.sna can be used to coerce data in any of the above forms to an sna edgelist. By turns, the function as.sociomatrix.sna can be used to convert any of these data types to adjacency matrix form.

To get started with sna, try obtaining viewing the list of available functions. This can be accomplished via the command library(help=sna).

Note

If you use this package and/or software manual in your work, a citation would be appreciated. References to the current versions are:

Butts, Carter T. (2016). "sna: Tools for Social Network Analysis." R package version 2.4.

Butts, Carter T. (2016). "Software Manual for the R sna Package." R package version 2.4.

See also the following paper, which explores the package in some detail:

Butts, Carter T. (2008). "Social Network Analysis with sna." *Journal of Statistical Software*, 24(6).

If utilizing a contributed routine, please also consider recognizing the author(s) of that specific function. Contributing authors, if any, are listed on the relevant manual pages. Your support helps to encourage the growth of the sna package, and is greatly valued!

Author(s)

Carter T. Butts <buttsc@uci.edu>

---

sna-coercion *sna Coercion Functions*

Description

Functions to coerce network data into one form or another; these are generally internal, but may in some cases be helpful to the end user.

Usage

```
as.sociomatrix.sna(x, attrname=NULL, simplify=TRUE, force.bipartite=FALSE)
## S3 method for class 'sna'
as.edgelist(x, attrname = NULL, as.digraph = TRUE,
    suppress.diag = FALSE, force.bipartite = FALSE, ...)
is.edgelist.sna(x)
```

Arguments

`x`	network data in any of several acceptable forms (see below).
`attrname`	if `x` is a `network` object, the (optional) edge attribute to be used to obtain edge values.
`simplify`	logical; should output be simplified by collapsing adjacency matrices of identical dimension into adjacency arrays?
`force.bipartite`	logical; should the data be interpreted as bipartite (with rows and columns representing different data modes)?
`as.digraph`	logical; should `network` objects be coded as digraphs, regardless of object properties? (Recommended)
`suppress.diag`	logical; should loops be suppressed?
`...`	additional arguments to `sna.edgelist` (currently ignored).

Details

The `sna` coercion functions are normally called internally within user-level `sna` functions to convert network data from various supported forms into a format usable by the function in question. With few (if any) exceptions, formats acceptable by these functions should be usable with any user-level function in the `sna` library.

`as.sociomatrix.sna` takes one or more input graphs, and returns them in adjacency matrix (and/or array) form. If `simplify==TRUE`, consolidation of matrices having the same dimensions into adjacency arrays is attempted; otherwise, elements are returned as lists of matrices/arrays.

`as.edgelist.sna` takes one or more input graphs, and returns them in `sna` edgelist form – i.e., a three-column matrix whose rows represent edges, and whose columns contain (respectively) the sender, receiver, and value of each edge. (Undirected graphs are generally assumed to be coded as fully mutual digraphs; edges may be listed in any order.) `sna` edgelists must also carry an attribute named `n` indicating the number of vertices in the graph, and may optionally contain the attributes `vnames` (carrying a vector of vertex names, in order) and/or `bipartite` (optionally, containing the number of row vertices in a two-mode network). If the bipartite attribute is present and non-false, vertices whose numbers are less than or equal to the attribute value are taken to belong to the first mode (i.e., row vertices), and those of value greater than the attribute are taken to belong to the second mode (i.e., column vertices). Note that the `bipartite` attribute is not strictly necessary to represent two-mode data, and may not be utilized by all `sna` functions.

`is.edgelist.sna` returns `TRUE` if its argument is a `sna` edgelist, or `FALSE` otherwise; if called with a list, this check is performed (recursively) on the list elements.

Data for `sna` coercion routines may currently consist of any combination of standard or sparse (via `SparseM`) adjacency matrices or arrays, `network` objects, or `sna` edgelists. If multiple items are given, they must be contained within a `list`. Where adjacency arrays are specified, they must be in three-dimensional form, with dimensions given in graph/sender/receiver order. Matrices or arrays having different numbers of rows and columns are taken to be two-mode adjacency structures, and are treated accordingly; setting `force.bipartite` will cause square matrices to be treated in similar fashion. In the case of `network` or `sna` edgelist matrices, bipartition information is normally read from the object's internal properties.

Value

An adjacency or edgelist structure, or a list thereof.

Note

For large, sparse graphs, edgelists can be dramatically more efficient than adjacency matrices. Where such savings can be realized, sna package functions usually employ sna edgelists as their "native" format (coercing input data with as.edgelist.sna as needed). For this reason, users of large graphs can often obtain considerable savings by storing data in edgelist form, and passing edgelists (rather than adjacency matrices) to sna functions.

The maximum size of adjacency matrices and edgelists depends upon R's vector allocation limits. On a 64-bit platform, these limits are currently around 4.6e4 vertices (adjacency case) or 7.1e8 edges (edgelist case). The number of vertices in the edgelist case is effectively unlimited (and can technically be infinite), although not all functions will handle such objects gracefully. (Use of vertex names will limit the number of edgelist vertices to around 2e9.)

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

sna, network

Examples

```
#Produce some random data, and transform it
g<-rgraph(5)
g
all(g==as.sociomatrix.sna(g))     #TRUE
as.edgelist.sna(g)                #View in edgelist form
as.edgelist.sna(list(g,g))        #Double the fun
g2<-as.sociomatrix.sna(list(g,g)) #Will simplify to an array
dim(g2)
g3<-as.sociomatrix.sna(list(g,g),simplify=FALSE)  #Do not simplify
g3                                                #Now a list

#We can also build edgelists from scratch...
n<-6
edges<-rbind(
c(1,2,1),
c(2,1,2),
c(1,3,1),
c(1,5,2),
c(4,5,1),
c(5,4,1)
)
attr(edges,"n")<-n
attr(edges,"vnames")<-letters[1:n]
gplot(edges,displaylabels=TRUE)               #Plot the graph
as.sociomatrix.sna(edges)                     #Show in matrix form
```

```
#Two-mode data works similarly
n<-6
edges<-rbind(
c(1,4,1),
c(1,5,2),
c(4,1,1),
c(5,1,2),
c(2,5,1),
c(5,2,1),
c(3,5,1),
c(3,6,2),
c(6,3,2)
)
attr(edges,"n")<-n
attr(edges,"vnames")<-c(letters[1:3],LETTERS[4:6])
attr(edges,"bipartite")<-3
edges
gplot(edges,displaylabels=TRUE,gmode="twomode")  #Plot
as.sociomatrix.sna(edges)                        #Convert to matrix
```

---

sna-deprecated *Deprecated Functions in sna Package*

Description

These functions are provided for compatibility with older versions of sna only, and may be defunct as soon as the next release.

Details

The following sna functions are currently deprecated:

None.

The original help pages for these functions can be found at help("oldName-deprecated"). Please avoid using them, since they will disappear....

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

Deprecated

sna.operators *Graphical Operators*

Description

These operators allow for algebraic manupulation of graph adjacency matrices.

Usage

```
## S3 method for class 'matrix'
e1 %c% e2
```

Arguments

e1	an (unvalued) adjacency matrix.
e2	another (unvalued) adjacency matrix.

Details

Currently, only one operator is supported. `x %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

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: University of Cambridge Press.

Examples

```
#Create an in-star
g<-matrix(0,6,6)
g[2:6,1]<-1
gplot(g)

#Compose g with its transpose
gcgt<-g%c%t(g)
gplot(gcgt,diag=TRUE)
gcgt
```

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 `<buttsc@uci.edu>`

References

Krackhardt, D. (1987). *Cognitive Social Structures*, 9, 109-134.

Examples

```
#Start with some random reports
g<-rgraph(10)

#Transform to CSS format
c<-sr2css(g)
```

stackcount — *How Many Graphs are in a Graph Stack?*

Description

Returns the number of graphs in the stack provided by `d`.

Usage

```
stackcount(d)
```

Arguments

`d`	a graph or graph stack.

Value

The number of graphs in `d`

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

`nties`

Examples

```
stackcount(rgraph(4,8))==8
```

stresscent — *Compute the Stress Centrality Scores of Network Positions*

Description

`stresscent` takes one or more graphs (`dat`) and returns the stress centralities of positions (selected by `nodes`) within the graphs indicated by `g`. Depending on the specified mode, stress 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

```
stresscent(dat, g=1, nodes=NULL, gmode="digraph",
    diag=FALSE, tmaxdev=FALSE, cmode="directed",
    geodist.precomp=NULL, rescale=FALSE, ignore.eval=TRUE)
```

Arguments

- `dat` one or more input graphs.
- `g` Integer indicating the index of the graph for which centralities are to be calculated (or a vector thereof). 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.
- `ignore.eval` logical; should edge values be ignored when calculating density?

Details

The stress of a vertex, v, is given by

$$C_S(v) = \sum_{i,j:i \neq j,i \neq v,j \neq v} g_{ivj}$$

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, matrix, or list containing the centrality scores (depending on the number and size of the input graphs).

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

Shimbel, A. (1953). "Structural Parameters of Communication Networks." *Bulletin of Mathematical Biophysics,* 15:501-507.

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=NULL, g2=NULL, 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=NULL)
```

Arguments

`dat`	one or more input graphs.
`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

$$d_S\left(G, H \left| L_G, L_H\right.\right) = \min_{L_G, L_H} d\left(\ell\left(G\right), \ell\left(H\right)\right)$$

where L_G is the set of accessible permutations/labelings of G, and $\ell(G)$ is a permuation/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 `cbind`ing two copies together. If `exchange.list` is already an nx2 matrix, it is left as-is. Once the nx2 exchangeabiliy 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 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 <buttsc@uci.edu>

References

Butts, C.T. and Carley, K.M. (2005). “Some Simple Algorithms for Structural Comparison.” *Computational and Mathematical Organization Theory,* 11(4), 291-305.

Butts, C.T., and Carley, K.M. (2001). “Multivariate Methods for Interstructural Analysis.” CASOS Working Paper, Carnegie Mellon University.

See Also

hdist, sdmat

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?
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)
```

`structure.statistics` *Compute Network Structure Statistics*

Description

Computes the structure statistics for the graph(s) in `dat`.

Usage

```
structure.statistics(dat, geodist.precomp = NULL)
```

Arguments

`dat`	one or more input graphs.
`geodist.precomp`	a `geodist` object (optional).

Details

Let $G = (V, E)$ be a graph of order N, and let $d(i, j)$ be the geodesic distance from vertex i to vertex j in G. The "structure statistics" of G are then given by the series $s_0, \ldots, s_{N-1}$, where $s_i = \frac{1}{N^2} \sum_{j \in V} \sum_{k \in V} I(d(j, k) \leq i)$ and I is the standard indicator function. Intuitively, s_i is the expected fraction of G which lies within distance `i` of a randomly chosen vertex. As such, the structure statistics provide an index of global connectivity.

Structure statistics have been of particular importance to biased net theorists, because of the link with Rapoport's original tracing model. They may also be used along with component distributions or connectedness scores as descriptive indices of connectivity at the graph-level.

Value

A vector, matrix, or list (depending on `dat`) containing the structure statistics.

Note

The term "structure statistics" has been used somewhat loosely in the literature, a trend which seems to be accelerating. Users should carefully check references before comparing results generated by this routine with those appearing in published work.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Fararo, T.J. (1981). "Biased networks and social structure theorems. Part I." *Social Networks,* 3, 137-159.

Fararo, T.J. (1984). "Biased networks and social structure theorems. Part II." *Social Networks,* 6, 223-258.

Fararo, T.J. and Sunshine, M.H. (1964). "A study of a biased friendship net." Syracuse, NY: Youth Development Center.

See Also

geodist, component.dist, connectedness, bn

Examples

```
#Generate a moderately sparse Bernoulli graph
g<-rgraph(100,tp=1.5/99)

#Compute the structure statistics for g
ss<-structure.statistics(g)
plot(0:99,ss,xlab="Mean Coverage",ylab="Distance")
```

---

summary.bayes.factor *Detailed Summaries of Bayes Factor Objects*

Description

Returns a bayes.factor summary object.

Usage

```
## S3 method for class 'bayes.factor'
summary(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

`summary.bbnam` *Detailed Summaries of bbnam Objects*

Description

Returns a bbnam summary object

Usage

```
## S3 method for class 'bbnam'
summary(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

```
## S3 method for class 'blockmodel'
summary(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

---

summary.cugtest *Detailed Summaries of cugtest Objects*

Description

Returns a cugtest summary object

Usage

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

Arguments

object	An object of class cugtest
...	Further arguments passed to or from other methods

Value

An object of class summary.cugtest

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

cugtest

summary.lnam *Detailed Summaries of lnam Objects*

Description

Returns a lnam summary object.

Usage

```
## S3 method for class 'lnam'
summary(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

```
## S3 method for class 'netcancor'
summary(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

---

summary.netlm *Detailed Summaries of netlm Objects*

Description

Returns a netlm summary object

Usage

```
## S3 method for class 'netlm'
summary(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

summary.netlogit *Detailed Summaries of netlogit Objects*

Description

Returns a `netlogit` summary object~

Usage

```
## S3 method for class 'netlogit'
summary(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

summary.qaptest *Detailed Summaries of qaptest Objects*

Description

Returns a `qaptest` summary object

Usage

```
## S3 method for class 'qaptest'
summary(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`

`symmetrize` *Symmetrize an Adjacency Matrix*

Description

Symmetrizes the elements of `mats` according to the rule in `rule`.

Usage

```
symmetrize(mats, rule="weak", return.as.edgelist=FALSE)
```

Arguments

`mats`	a graph or graph stack.
`rule`	one of “upper”, “lower”, “strong” or “weak”.
`return.as.edgelist`	logical; should the symmetrized graphs be returned in edgelist form?

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 <buttsc@uci.edu>

References

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.

Examples

```
#Generate a graph
g<-rgraph(5)

#Weak symmetrization
symmetrize(g)

#Strong symmetrization
symmetrize(g,rule="strong")
```

`triad.census` *Compute the Davis and Leinhardt Triad Census*

Description

`triad.census` returns the Davis and Leinhardt triad census of the elements of `dat` indicated by `g`.

Usage

```
triad.census(dat, g=NULL, mode = c("digraph", "graph"))
```

Arguments

`dat`	a graph or graph stack.
`g`	the elements of `dat` to process.
`mode`	string indicating the directedness of edges; `"digraph"` implies a directed structure, whereas `"graph"` implies an undirected structure.

Details

The Davis and Leinhardt triad census consists of a classification of all directed 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.

In the undirected case, the triad census reduces to four states (based on the number of edges in each triad. Where `mode=="graph"`, this is returned instead.

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, in the directed case. In the undirected case, only 4 columns are used.

Warning

Valued data may cause strange behavior with this routine. Dichotomize the data first.

Author(s)

Carter T. Butts <buttsc@uci.edu>

References

Davis, J.A. and Leinhardt, S. (1972). "The Structure of Positive Interpersonal Relations in Small Groups." In J. Berger (Ed.), *Sociological Theories in Progress, Volume 2*, 218-251. Boston: Houghton Mifflin.

Wasserman, S., and Faust, K. (1994). "Social Network Analysis: Methods and Applications." Cambridge: Cambridge University Press.

See Also

`triad.classify`, `dyad.census`, `kcycle.census`, `kpath.census`, `gtrans`

Examples

```
#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 `g`th graph of stack `dat`.

Usage

```
triad.classify(dat, g=1, tri=c(1, 2, 3), mode=c("digraph",
    "graph"))
```

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.
`mode`	string indicating the directedness of edges; `"digraph"` implies a directed structure, whereas `"graph"` implies an undirected structure.

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` $a \to b \not\leftrightarrow c, a \not\leftrightarrow c$

`102` $a \leftrightarrow b \not\leftrightarrow c, a \not\leftrightarrow c$

`021D` $a \leftarrow b \to c, a \not\leftrightarrow c$

`021U` $a \to b \leftarrow c, a \not\leftrightarrow c$

`021C` $a \to b \to c, a \not\leftrightarrow c$

`111D` $a \not\leftrightarrow b \to c, a \leftrightarrow c$

`111U` $a \not\leftrightarrow b \leftarrow c, a \leftrightarrow c$

`030T` $a \to b \leftarrow c, a \to c$

`030C` $a \leftarrow b \leftarrow c, a \to c$

`201` $a \leftrightarrow b \not\leftrightarrow c, a \leftrightarrow c$

`120D` $a \leftarrow b \to c, a \leftrightarrow c$

`120U` $a \to b \leftarrow c, a \leftrightarrow c$

`120C` $a \to b \to c, a \leftrightarrow c$

`210` $a \to b \leftrightarrow c, a \leftrightarrow c$

`300` $a \leftrightarrow b \leftrightarrow c, a \leftrightarrow c$

These codes are returned by `triad.classify` as strings. In the undirected case, only four triad states are possible (corresponding to the number of edges in the triad). These are evaluated for `mode=="graph"`, with the return value being the number of edges.

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 <buttsc@uci.edu>

References

Davis, J.A. and Leinhardt, S. (1972). "The Structure of Positive Interpersonal Relations in Small Groups." In J. Berger (Ed.), *Sociological Theories in Progress, Volume 2*, 218-251. Boston: Houghton Mifflin.

Wasserman, S., and Faust, K. (1994). *Social Network Analysis: Methods and Applications.* Cambridge: Cambridge University Press.

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 or 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>

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)
```

---

write.dl *Write Output Graphs in DL Format*

Description

Writes a graph stack to an output file in DL format.

Usage

```
write.dl(x, file, vertex.lab = NULL, matrix.lab = NULL)
```

Arguments

x	a graph or graph stack, of common order.
file	a string containing the filename to which the data should be written.
vertex.lab	an optional vector of vertex labels.
matrix.lab	an optional vector of matrix labels.

Details

DL format is used by a number of software packages (including UCINET and Pajek) to store network data. `write.dl` saves one or more (possibly valued) graphs in DL edgelist format, along with vertex and graph labels (if desired). These files can, in turn, be used to import data into other software packages.

Value

None.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

write.nos

Examples

```
## Not run:
#Generate a random graph stack
g<-rgraph(5,10)

#This would save the graphs in DL format
write.dl(g,file="testfile.dl")

## End(Not run)
```

---

write.nos *Write Output Graphs in (N)eo-(O)rg(S)tat Format*

Description

Writes a graph stack to an output file in NOS format.

Usage

```
write.nos(x, file, row.col = NULL, col.col = NULL)
```

Arguments

x	a graph or graph stack (all graphs must be of common order).
file	string containing the output file name.
row.col	vector of row labels (or "row colors").
col.col	vector of column labels ("column colors").

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). `write.nos` provides a quick and dirty way of writing files NOS, which can later be retrieved using `read.nos`.

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).

Note that NOS format only supports graph stacks of common order; graphs of different sizes cannot be stored within the same file.

Value

None.

Author(s)

Carter T. Butts <buttsc@uci.edu>

See Also

read.nos, write.dl, write.table

Examples

```
## Not run:
#Generate a random graph stack
g<-rgraph(5,10)

#This would save the graphs in NOS format
write.nos(g,file="testfile.nos")

#We can also read them back, like so:
g2<-read.nos("testfile.nos")

## End(Not run)
```

Index