Package 'SANTA'

April 10, 2023

Type Package

Title Spatial Analysis of Network Associations

Version 2.34.0

Date 6 October 2021

Imports graphics, Matrix, methods, stats

Depends R (>= 4.1), igraph

Suggests BiocGenerics, BioNet, DLBCL, formatR, knitr, msm, org.Sc.sgd.db, markdown, rmarkdown, RUnit

Description This package provides methods for measuring the strength of association between a network and a phenotype. It does this by measuring clustering of the phenotype across the network (Knet). Vertices can also be individually ranked by their strength of association with high-weight vertices (Knode).

License GPL (>= 2)

LazyLoad yes

VignetteBuilder knitr

biocViews Network, NetworkEnrichment, Clustering

BugReports https://github.com/alexjcornish/SANTA

git_url https://git.bioconductor.org/packages/SANTA

git_branch RELEASE_3_16

git_last_commit 827739a

git_last_commit_date 2022-11-01

Date/Publication 2023-04-10

Author Alex Cornish [cre, aut]

Maintainer Alex Cornish <alex.cornish@icr.ac.uk>

R topics documented:

`BinGraph` *Compute a distance bin matrix from a distance matrix*

Description

In order for the `Knet` or `Knode` functions to be run on a network, it is necessary to place the raw distances between each vertex pair into discreet distance bins, as the functions are unable to handle continuous distributions of distances. This is done automatically within the `Knet` and `Knode` functions, but can also be done separately using `BinGraph`.

Usage

```
BinGraph(D, nsteps=1000, equal.bin.fill=TRUE, verbose=TRUE)
```

Arguments

`D`	Numeric matrix, a distance matrix output by `DistGraph`.
`nsteps`	Integer value, the desired number of bins across which the distances are to be split. If there are too few unique distances to fill each bin, then fewer bins are returned.
`equal.bin.fill`	Logical, if `TRUE` then the function attempts to fill each bin with an equal number of vertex pairs.
`verbose`	Logical, if `TRUE` messages about the progress of the function are displayed.

Details

In order for the `Knet` or `Knode` functions to be run, the vertex pair distances (as computed by `DistGraph`) but be split into bins. This is done as part of the `Knet` or `Knode`. However, this step is often slow for large networks and therefore the `BinGraph` function is provided separately, in order to avoid repeat computation.

Each vertex pair is placed into a bin, either ranging from 1 to `nsteps`, or from 1 to the number of unique distances.

If `equal.bin.fill` is `FALSE`, then the bin each vertex pair is placed into is directly proportional to the largest vertex pair distance. For example, if the distance between the pair is 25% of the largest distance and `nsteps` equals `100`, then the vertex pair will be placed into bin `25`. However, this can

create problems when there are a small number of edges with especially large distances, as this can result in the majority of vertex pairs being placed into a small number of bins. This can reduce the effectiveness of the Knet and Knode functions. Therefore, when equal.bin.fill is TRUE, the function attempts to fill each bin with an equal number of vertex pairs. If there are a large number of tied distances, then the bins may not be filled equally.

Value

Integer matrix with the same dimensions as D.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

See Also

DistGraph

Examples

```
# create network and calculate the distance matrix using the shortest paths measure
g1 <- barabasi.game(6, directed=FALSE)
plot(g1, layout=layout.fruchterman.reingold)
D1 <- DistGraph(g1, dist.method="shortest.paths")
# place the distances into distance bins
BinGraph(D1, nsteps=100)

# create network and calculate the distance matrix using diffusion kernel-based measure
g2 <- erdos.renyi.game(6, p.or.m=0.5, directed=FALSE)
g2 <- set.edge.attribute(g2, name="distance", value=runif(ecount(g2)))
plot(g2, layout=layout.fruchterman.reingold)
# place the distances into distance bins
D2 <- DistGraph(g2, dist.method="diffusion", edge.attr="distance")
BinGraph(D2, nsteps=100)
```

Compactness *Measure the strength of association using compactness scores*

Description

The compactness score of set of hits on a network is the mean distance between each pair of hits. By comparing the observed compactness score to the scores of permuted hit sets, it is possible to compute the significance of the strength of association between the phenotype and the network. This method is not as effective as the Knet function and is included only for comparison.

Usage

```
Compactness(g, nperm=100, dist.method=c("shortest.paths", "diffusion", "mfpt"),
vertex.attr="pheno", edge.attr="distance", correct.factor=1, D=NULL,
verbose=TRUE)
```

Arguments

g	igraph object, the network to work on.
nperm	Integer value, the number of permutations to be completed.
dist.method	String, the method used to compute the distance between each pair of hits on the network.
vertex.attr	Character vector, the name of the vertex attributes under which the hits to be tested are stored. The vector can contain one or more vertex attributes.
edge.attr	String, the name of the edge attribute to be used as distances along the edges. If an edge attribute with this name is not found, then each edge is assumed to have a distance of 1. Smaller edge distances denote stronger interactions between vertex pairs.
correct.factor	Numeric value. If the network contains unconnected vertices, then the distance between these vertices is set as the maximum distance between the connected vertices multiplied by correct.factor.
D	Symmetrical numerical matrix. A precomputed distance matrix for g output by the DistGraph function. If NULL, then D is computed by the Compactness function.
verbose	Logical, if TRUE messages about the progress of the function are displayed.

Details

The compactness score is used by the PathExpand tool by Glaab et al. (2010). It is a measure of the mean distance between a set of genes in a network. By comparing the compactness score of an observed set of hits to sets of permuted hits, it is possible to produce a p-value describing the strength of association between the gene set and the network. This is not some done within the original paper by Glaab et al. (2010). The function is much like the Knet function, albeit not as effective.

The compactness score C is defined as the mean shortest path distance between pairs of vertices in a set P on network g.

$$C(P) = \frac{2\sum_{i,j \in P; i<j} d^g(i,j)}{|P| * (|P| - 1)}$$

The compactness score is only included within the SANTA package to allow for comparisons to be made. Unlike the Knet function, it cannot be applied to continuous distributions of vertex weights. It can also result in biases if there is large variability in density across the network.

The weight of a vertex should be 1 if it is a hit, 0 if it is not a hit or NA if the information is missing. Vertices with missing weights are still included within the network but are excluded from the permuted sets.

Value

If one vertex attribute is input, Compactness is run on the single set of vertex weights and a list containing the statistics below is returned. If more than one vertex attribute is input, then Compactness is run on each set of vertex weights and a list containing an element for each vertex attribute is returned. Each element contains a sub-list containing the statistics below for the relevant vertex attribute.

score.obs	Observed compactness score
score.perm	Permuted compactness scores. NA if no permutations are completed.
pval	p-value, computed from a z-score derived from the observed and permuted compactness scores. NA if no permutations are completed.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

References

Cornish, A.J. and Markowetz, F. (2014) *SANTA: Quantifying the Functional Content of Molecular Networks.*. PLOS Computational Biology. 10:9, e1003808.

Glaab, E., Baudot A., Krasnogor N. and Valencia A. (2010). *Extending pathways and processes using molecular interaction networks to analyse cancer genome data*. BMC Bioinformatics. 11(1): 597:607.

CreateGrid *Generate a grid-like network*

Description

Generate a network with a grid-like arrangement of edges.

Usage

```
CreateGrid(n=100)
```

Arguments

n	Integer value, the number of vertices to be included.

Details

This is a simple algorithm that creates a grid-like network. Vertices are arranged in the largest square lattice possible. Vertices not included within this square are added as an additional row. Vertices are connected by edges to their closest neighbours.

Value

igraph object.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

Examples

```
# generate and plot a grid-like network containing 100 vertices.
g <- CreateGrid(n = 100)
plot(g, layout=layout.fruchterman.reingold)
```

DistGraph *Compute the vertex pair distance matrix for a graph*

Description

Compute the distances between pairs of vertices in a graph, using a shortest path, diffusion kernel, or mean first-passage time-based measure.

Usage

```
DistGraph(g, v=V(g), edge.attr=NULL, dist.method=c("shortest.paths", "diffusion", "mfpt"),
correct.inf=TRUE, correct.factor=1, verbose=TRUE)
```

Arguments

g	igraph object, the graph on which to work.
v	igraph object or numeric vector, the vertices from which each distance is calculated.
edge.attr	String, the name of the edge attribute to be used as distances along the edges. If NULL, then each edge is assumed to have a distance of 1. Smaller edge distances denote stronger interactions between vertex pairs.
dist.method	String, the method used to compute the distance between each vertex pair.
correct.inf	Logical, if TRUE then infinite vertex pair distances are replaced with distances equal to the maximum distance measured across the network, multiplied by correct.factor.
correct.factor	Numeric value, if the graph contains unconnected vertices, then the distance between these vertices is set as the maximum distance between the connected vertices multiplied by correct.factor.
verbose	Logical, if TRUE messages about the progress of the function are displayed.

Details

This function computes a distance matrix for a graph. Different methods can be used to calculate the distance between each pair of vertices. If a set of vertices is specified, a smaller distance matrix containing only vertices corresponding to the vertices is returned.

Descriptions of how the shortest paths (shortest.paths), diffusion kernel-based (GraphDiffusion) and mean first-passage time (GraphMFPT) distance measures work are given in their respective function descriptions.

Value

Numeric matrix, containing the distances between vertex pairs.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

References

Kondor, R.I. and Lafferty, J. (2002). *Diffusion Kernels on Graph and Other Discrete Structures*. Proc. Intl. Conf. Machine Learning.

White, S. and Smyth, P. (2003). *Algorithms for Estimating Relative Importance in Networks*. Technical Report UCI-ICS 04-25.

See Also

shortest.paths, GraphDiffusion, GraphMFPT

Examples

```
# create a graph and compute the distance matrix using the shortest paths measure
g1 <- barabasi.game(6, directed=FALSE)
DistGraph(g1, dist.method="shortest.paths")
plot(g1, layout=layout.fruchterman.reingold)

# create a graph, assign edge distances and compute the distance matrix using the
# diffusion kernel-based measure
g2 <- erdos.renyi.game(6, p.or.m=0.5, directed=FALSE)
g2 <- set.edge.attribute(g2, name="distance", value=runif(ecount(g2)))
DistGraph(g2, dist.method="diffusion", edge.attr="distance")
plot(g2, layout=layout.fruchterman.reingold)
```

GraphDiffusion *Compute diffusion kernel-based distance matrix*

Description

Using a diffusion kernel-based algorithm, compute the distance between vertex pairs in an undirected network, with or without edge weights. This algorithm provides an alternative to the shortest.paths and mfpt measures of vertex pair distance.

Usage

```
GraphDiffusion(g, v=V(g), edge.attr.weight=NULL, beta=1, correct.neg=TRUE)
```

Arguments

g	igraph object, the network to work on.
v	igraph object or numeric vector, the vertices from which each distance is calculated.
edge.attr.weight	String, the name of the edge attribute to be used as weights along the edges. Greater weights indicate a stronger interaction between the two genes (this is the opposite to edge distances, where smaller distances indicate stronger interactions). If NULL, then each edge is assumed to have a weight of 1.
beta	Numeric value, the probability that the diffusion process will take an edge emanating from a vertex.
correct.neg	Logical, if TRUE then negative edge distances are set to 0.

Details

Diffusion across a network follows a process similar to a random walk. This provides a method of measuring the distance between vertex pairs that does not simply take into account a single path (like the shortest.paths algorithm) but instead incorporates multiple paths. This function uses a diffusion kernel-based approach to compute distances. The algorithm implemented is detailed in the referenced paper.

The distance from vertex A to vertex A is always 0.

Value

Numeric matrix, containing the diffusion kernel-based vertex pair distances between each vertex in v and every vertex in g.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

References

Kondor, R.I. and Lafferty, J. (2002). *Diffusion Kernels on Graph and Other Discrete Structures*. Proc. Intl. Conf. Machine Learning.

See Also

GraphMFPT, shortest.paths

Examples

```
# create a network and computes the diffusion kernel-derived vertex pair distance matrix
g <- barabasi.game(6, directed=FALSE)
GraphDiffusion(g)
plot(g, layout=layout.fruchterman.reingold)
```

GraphMFPT *Compute mean first-passage time-based distance matrix*

Description

Using the mean first-passage time method, compute the distances between vertex pairs in an undirected graph, with or without edge weights.

Usage

```
GraphMFPT(g, v=V(g), edge.attr.weight=NULL, average.distances=TRUE)
```

Arguments

g	igraph object, the graph to work on.
v	igraph object or numeric vector, the vertices from which each distance is calculated.
edge.attr.weight	String, the name of the edge attribute to be used as weights along the edges. Greater weights indicate a stronger interaction between the two genes (this is the opposite to edge distances, where smaller distances indicate stronger interactions). If NULL then each edge is assumed to have a weight of 1.
average.distances	Logical, if TRUE then the distance from vertex A to B and the distance from vertex B to A are averaged to give a single distance. Otherwise, two different distances may be returned.

Details

The mean first-passage time from vertex A to vertex B is defined as the expected number of steps taken on a random walk emanating from vertex A until the first arrival at vertex B. This provides a method of measuring the distance between pairs of vertices that does not simply take into account the distance along the shortest path, but rather incorporates how well the two vertices are connected across multiple paths.

The mean first-passage time from vertex A to vertex B is not necessarily the same as the mean first-passage time from vertex B to vertex A. If a symmetric distance matrix is required, reciprocal distances can be averaged to give a single value for each vertex pair.

If a vertex pair is unconnected, then the distance between the vertices is Inf.

The distance from vertex A to vertex A is always 0.

Value

Numeric matrix, containing the mean first-passage time-derived vertex pair distance between each vertex in v and every vertex in g.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

References

White, S. and Smyth, P. (2003). *Algorithms for Estimating Relative Importance in Networks*. Technical Report UCI-ICS 04-25.

See Also

GraphDiffusion, shortest.paths

Examples

```
# create a  and compute the mean first-passage time-based vertex pair distance matrix
g <- erdos.renyi.game(6, p.or.m=0.5, directed=FALSE)
GraphMFPT(g)
plot(g, layout=layout.fruchterman.reingold)
```

Knet — *Measure the strength of association between a phenotype and a network by computing the strength of hit clustering on the network*

Description

Compute the strength of clustering of high-weight vertices (hits) on a network using a modified version of Ripley's K-statistic. This method can be used to measure the strength of association between a phenotype or function and a network.

Usage

```
Knet(g, nperm=100, dist.method=c("shortest.paths", "diffusion", "mfpt"),
vertex.attr="pheno", edge.attr=NULL, correct.factor=1, nsteps=1000,
prob=c(0, 0.05, 0.5, 0.95, 1), B=NULL, verbose=TRUE)
```

Arguments

g	igraph object, the network to work on.
nperm	Integer value, the number of permutations to be completed.
dist.method	String, the method used to calculate the distance between vertex pairs.
vertex.attr	Character vector, the name of the vertex attributes under which the vertex weights to be tested are stored. The vector can contain one or more elements.
edge.attr	String, the name of the edge attribute to be used as distances along the edges. If an edge attribute with this name is not found, then each edge is assumed to have a distance of 1.

correct.factor	Numeric value, if the network contains unconnected vertices, then the distance between these vertices is set as the maximum distance between the connected vertices multiplied by correct.factor.
nsteps	Integer value, the number of bins into which vertex pairs are placed.
prob	Numeric vector, the quantiles to be calculated for the Knet permutations.
B	Symmetrical numeric matrix. A precomputed distance bin matrix for g output by the BinGraph function. If NULL, then B is computed within the Knet function.
verbose	Logical, if TRUE messages about the progress of the function are displayed.

Details

The SANTA method uses the 'guilt-by-association' principle to measure the strength of association between a network and a phenotype. It does this by measuring the strength of clustering of the phenotype scores across the network. The stronger the clustering, the greater the association between the network and the phenotype.

The SANTA method applies Ripley's K-function, a well-established approach to spatial statistics that measures the strength of clustering of points on a plane, and extends it in a number of ways. First, a Knet function is defined by adapting the approach for networks using vertex pair distance measures. Second, vertex weights are incorporated into Knet and the importance of vertices made relative to their own associated weight. Third, the mean vertex weight is subtracted from each individual vertex weight when calculating the Knet function. This means that the Knet function measures the degree of vertex weight clustering relative to a random distribution of vertex weights. The Knet function is defined as

$$K^{net}[s] = \frac{2}{p^2} \sum_i p_i \sum_j (p_j - \bar{p}) I(d^g[i,j] <= s)$$

where p_i is the weight of vertex i, $\bar{p}$ is the mean vertex weight across all vertices, and $I(dg[i,j] <= s)$ is an identity function, equaling 1 if vertex i and vertex j are within distance s and 0 otherwise.

In order to derive a p-value and quantify the significance of the observed distribution of weights, the observed Knet-curve is compared to Knet-curves obtained using the same network but randomly permuted vertex weights. Vertices with missing weights (NA) are not included within these permutations. The area under the Knet-curve (AUK) is calculated for the observed network and each of the permuted networks and a z-score used to produce a p-value. This p-value indicates the probability an observed AUK at least this high is seen given the null hypothesis that the vertex weights are randomly distributed.

Vertex weights should be greater or equal that zero or equal to NA if the weight is missing.

Value

If one vertex attribute is input, Knet is run on the single set of vertex weights and a list containing the statistics below is returned. If more than one vertex attribute is input, then Knet is run on each set of vertex weights and a list containing an element for each vertex attribute is returned. Each element contains a sub-list containing the statistics below for the relevant vertex attribute.

K.obs	Knet-function curve for the observed vertex weights.
AUK.obs	Area under the Knet-function curve (AUK) for the observed vertex weights.

K.perm	Knet-function curve for each permutation of vertex weights. Equals NA if no permutations are completed.
AUK.perm	Area under the Knet-function curve (AUK) for each permutation of vertex weights. NA if no permutations are completed.
K.quan	Quantiles for the permuted Knet-function curves. NA if no permutations are completed.
pval	p-value, calculated from a z-score derived from the observed and permuted AUKs. NA if no permutations are completed.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk> and Florian Markowetz

References

Cornish, A.J. and Markowetz, F. (2014) *SANTA: Quantifying the Functional Content of Molecular Networks.*. PLOS Computational Biology. 10:9, e1003808.

Okabe, A. and Yamada, I. (2001). *The K-function method on a network and its computational implementation* Geographical Analysis. 33(3): 271-290.

See Also

Knode

Examples

```
# apply Knet to a network with hit clustering
g.clustered <- barabasi.game(50, directed=FALSE)
g.clustered <- SpreadHits(g.clustered, h=10, lambda=10)
res.clustered <- Knet(g.clustered, nperm=100, vertex.attr="hits")
res.clustered$pval
plot.Knet(res.clustered)

# apply Knet to a network without hit clustering
g.unclustered <- barabasi.game(50, directed=FALSE)
g.unclustered <- SpreadHits(g.unclustered, h=10, lambda=0)
res.unclustered <- Knet(g.unclustered, nperm=100, vertex.attr="hits")
res.unclustered$pval
plot.Knet(res.unclustered)
```

Knode *Rank vertices by their strength of association with high-weight vertices*

Description

Rank vertices by their strength of association with high-weight vertices using a modified version of Ripley's K-statistic. Vertex weights can either be binary or positive and continuous.

Usage

```
Knode(g, dist.method=c("shortest.paths", "diffusion", "mfpt"), vertex.attr="pheno",
edge.attr=NULL, correct.factor=1, nsteps=1000, B=NULL, verbose=TRUE)
```

Arguments

g	igraph object, the network to work on.
dist.method	String, the method used to calculate the distance between each vertex pair.
vertex.attr	Character vector, the name of the vertex attributes under which the vertex weights to be tested are stored. The vector can contain one or more elements.
edge.attr	String, the name of the edge attribute to be used as distances along the edges.
correct.factor	Numeric value, if the network contains unconnected vertices, then the distance between these vertices is set as the maximum distance between the connected vertices multiplied by correct.factor.
nsteps	Integer value, the number of bins into which vertex pairs are placed.
B	Symmetric numerical matrix. A precomputed distance bin matrix for g output by the BinGraph function. If NULL, then B is computed within the Knode function.
verbose	Logical, if TRUE messages about the progress of the function are displayed.

Details

Using the inner sum of the Knet equation, it becomes possible to prioritise vertices by how well they are connected, or associated, with high-weight vertices. The inner sum of the Knet equation is

$$K_i^{node}[s] = \frac{2}{p}\sum_j (p_j - \bar{p}) I(d^g(i,j) <= s)$$

where p_j is the weight of vertex j, $\bar{p}$ is the mean vertex weight across all vertices, and $I(dg[i,j] <= s)$ is an identity function, equaling 1 if vertex i and vertex j are within distance s and 0 otherwise.

If the name of each vertex is stored within a vertex attribute called name, then the returned scores are labelled with these names.

Vertex weights should be greater than or equal to 0, or equal to NA if the weight is missing. The Knode statistic is still computed for vertices with missing weights.

If an edge attribute with this name is not found, then each edge is assumed to have a distance of 1. Smaller edge distances denote stronger interactions between vertex pairs

Value

A sorted named numerical vector of Knode AUKs for each vertex.

If one vertex attribute is input, then the Knode AUKs are calculated and a single numerical vector is returned. If more than one vertex attribute is input, then a list of vectors, one for each set of vertex weights, is returned.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk> and Florian Markowetz

References

Cornish, A.J. and Markowetz, F. (2014) *SANTA: Quantifying the Functional Content of Molecular Networks.*. PLOS Computational Biology. 10:9, e1003808.

See Also

Knet

Examples

```
# create a network with a single cluster of high-weight vertices
# rank all vertices by their strength of association with the high-weight vertices
g1 <- erdos.renyi.game(15, p.or.m=0.3, directed=FALSE)
g1 <- SpreadHits(g1, h=3, lambda=10)
Knode(g1, vertex.attr="hits")
plot(g1)
```

---

plot.Knet *Plot the results of the* Knet *function*

Description

Plot the observed Knet curve against the quantiles of the permuted Knet curves and the observed AUK against the permuted AUKs.

Usage

```
## S3 method for class 'Knet'
plot(x, sequential=FALSE, ...)
```

Arguments

x	Results from the Knet function.
sequential	Logical, if TRUE then the plots are sequential. Otherwise, the two plots are plotted alongside each other.
...	Additional arguments to be passed to plot.

Details

If the high-weight vertices are clustered, then the observed Knet curve and AUK will be high relative to the permuted Knet curves and AUKs. The greater the strength of clustering, the greater the difference between the observed and permuted statistics. If the strength of clustering is low, then the observed and permuted curves and AUKs will likely overlap.

The first plot displays the the observed curve in red and the quantiles of the permuted curves in yellow. The quantile boundaries are displayed as grey lines. These boundaries are specified in the Knet function. The second plot displays the observed AUK as a red line and the distribution of permuted AUKs in grey.

Value

Plots described in details section.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk> and Florian Markowetz

References

Cornish, A.J. and Markowetz, F. (2014) *SANTA: Quantifying the Functional Content of Molecular Networks.*. PLOS Computational Biology. 10:9, e1003808.

See Also

Knet

Examples

```
# plot results with hit clustering
g.clustered <- barabasi.game(100, directed=FALSE)
g.clustered <- SpreadHits(g.clustered, h=10, lambda=10)
res.clustered <- Knet(g.clustered, nperm=10, vertex.attr="hits")
res.clustered$pval
plot.Knet(res.clustered)

# plot results without hit clustering
g.unclustered <- barabasi.game(100, directed=FALSE)
g.unclustered <- SpreadHits(g.unclustered, h=10, lambda=0)
res.unclustered <- Knet(g.unclustered, nperm=10, vertex.attr="hits")
res.unclustered$pval
plot.Knet(res.unclustered)
```

SANTA.data *Pre-processed dataset for the SANTA vignette*

Description

Pre-processed network and expression data for use in the accompanying vignette.

Details

edgelist.humannet Data frame of functional interactions from the HumanNet network with edge distances derived from the accompanying log-likelihood scores.

edgelist.intact Data frame of *H. sapiens* physical interactions from the IntAct database downloaded on 2013-05-02.

g.bandyopadhyay.treated *S. cerevisiae* interaction network created from the MMS-treated GI data from the study by Bandyopadhyay et al. Interactions represent correlation in GI profile. Network is an igraph object.

g.bandyopadhyay.untreated *S. cerevisiae* interaction network created from the untreated GI data from the study by Bandyopadhyay et al. Interactions represent correlation in GI profile. Network is an `igraph` object.

g.costanzo.cor *S. cerevisiae* interaction network created from the GI data from the study by Costanzo et al. Interactions represent correlation in GI profile. Network is an `igraph` object.

g.costanzo.raw *S. cerevisiae* interaction network created from the GI data from the study by Costanzo et al. Interactions represent raw GIs. Network is an `igraph` object.

g.srivas.high *S. cerevisiae* interaction network created from the high UV-dosage GI data from the study by Srivas et al. Interactions represent raw GIs. Network is an `igraph` object.

g.srivas.untreated *S. cerevisiae* interaction network created from the untreated GI data from the study by Srivas et al. Interactions represent raw GIs. Network is an `igraph` object.

go.entrez Genes associated with the GO term GO:0042981 (regulation of apoptotic process).

rnai.cheung Matrix of gene-wise essentiality p-values for 6 cancer cell lines. Rows represent genes and columns represent cancers. This data was created from the RNAi screens conducted by Cheung et al. The weight of evidence approach was used to compute essentiality scores for each shRNA and GENE-E was used to collapse the shRNA scores into gene-wise scores.

References

Lee, I., Blom, U.M., Wang, P.I. et al. (2011). *Prioritizing candidate disease genes by network-based boosting of genome-wide association data.* Genome Research. 21, 1109-21.

Orchard, S., Ammari, M., Aranda, B. et al. (2014). *The MIntAct project - IntAct as a common curation platform for 11 molecular interaction databases.* Nucleic Acids Research. 42:1, D358-63.

Costanzo, M., Baryshnikova., A., Bellay, J. et al. (2010). *The Genetic Landscape of a Cell.* Science. 327:5964, 425-31.

Bandyopadhyay, S., Mehta, M., Kuo, D. et al. (2010) *Rewiring of Genetic Networks in Response to DNA Damage.* Science. 330, 1385-90.

Srivas, R., Costelloe, T., Carvunis, A. et al. (2013) *A UV-induced genetic network links the RSC complex to nucleotide excision repair and shows dose-dependant rewiring.* Cell Reports. 5:6, 1714-24.

Ashburner, M., Ball, C.A., Blake, J.A. et al. (2000) *Gene Ontology: tool for the unification of biology.* Nature Genetics. 25, 25-9.

Cheung, H.W, Cowley, G.S., Weir, B.A. et al. (2011) *Systematic investigation of genetic vulnerabilities across cancer cell lines reveals lineage-specific dependencies in ovarian cancer.* PNAS. 108:30, 12372-7.

Examples

```
data(g.bandyopadhyay.treated)
```

SpreadHits *Spread hits across a network in clusters*

Description

Spread hits across a network in one or more clusters using an exponential probability distribution related to the distance of each vertex from a seed vertex.

Usage

```
SpreadHits(g, h, clusters=1, distance.cutoff=3, lambda=1,
dist.method=c("shortest.paths", "diffusion", "mfpt"),
edge.attr=NULL, hit.color="red", D=NULL, attempts=1000, verbose=TRUE)
```

Arguments

g	igraph object, the network to work on.
h	Integer value, the number of hits in each cluster.
clusters	Integer value, the number of clusters to add the network.
distance.cutoff	Integer value, the minimum distance between the seed vertex of each cluster.
lambda	Numeric value, the strength of hit clustering. If lambda == 0, then the hits are randomly distributed. If lambda > 0, then the hits are clustered. The greater the value of lambda, the greater the strength of the hit clustering.
dist.method	String, the method used to compute the distance between each vertex and the seed vertex.
edge.attr	String, the name of the edge attribute to be used as distances along the edges. If NULL, then each edge is assumed to have a distance of 1. Smaller edge distances denote stronger interactions between vertex pairs.
hit.color	String, the colour of the hits if the network is plotted.
D	Pre-computed numeric distance matrix. If NULL, then D is computed within the function.
attempts	Integer value, the number of attempts made at each stage of the placement algorithm before the function terminates.
verbose	Logical, if TRUE messages about the progress of the function are displayed.

Details

SpreadHits can be used to add hits to a network without hits, or replace hits on a network with hits. The probability of a vertex being labelled as a hit depends on its distance from a randomly chosen starting vertex. The value of lambda denotes the shape of the probability distribution used to spread the hits. The greater the value of lambda, the greater the strength of hit clustering. The probability of vertex i being a hit is proportional to

$$P[i] \sim lambda^{-d[start,i]}$$

where $d[start, i]$ is the distance between the start vertex and vertex i.

Hits can be added in one or more clusters. If multiple clusters are being added, then the function first identifies an equal number of seed vertices at least `distance.cutoff` distance apart in the network. If the function cannot identify vertices that satisfy this condition within the specified number of attempts, then it returns `NULL`. If the function is able to find suitable seed vertices, then it proceeds to label vertices positioned around each seed vertex as hits, using a probability function shaped by `lambda`.

Whether a vertex is a hit or a miss is given in the vertex attribute `hits`. If the vertex is a hit, then it has the score 1. If it is a miss, then it has the score 0. A color can also be chosen to highlight the hits when the network is plotted. Misses are automatically coloured grey.

Value

A modified version of the input `igraph` object. Whether a vertex is a hit or miss is given under the vertex attribute `hits`.

Author(s)

Alex J. Cornish <a.cornish12@imperial.ac.uk>

Examples

```
# create a network and add 1 cluster of 3 hits
g1 <- erdos.renyi.game(30, p.or.m=0.1, directed=FALSE)
g1 <- SpreadHits(g1, h=3, clusters=1, lambda=10)
plot(g1, layout=layout.fruchterman.reingold)

# create a network and add 2 clusters of 3 hits
g2 <- erdos.renyi.game(30, p.or.m=0.1, directed=FALSE)
g2 <- SpreadHits(g2, h=3, clusters=3, distance.cutoff=2, lambda=0)
plot(g2, layout=layout.fruchterman.reingold)
```

Index