Package 'BioNAR'

November 11, 2026

Title Biological Network Analysis in R

Version 1.14.0

Description the R package BioNAR, developed to step by step analysis of PPI network. The aim is to quantify and rank each protein's simultaneous impact into multiple complexes based on network topology and clustering. Package also enables estimating of co-occurrence of diseases across the network and specific clusters pointing towards shared/common mechanisms.

License Artistic-2.0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 3.5.0), igraph (>= 2.0.1.1), poweRlaw, latex2exp, RSpectra, Rdpack

Imports stringr, viridis, fgsea, grid, methods, AnnotationDbi, dplyr, GO.db, org.Hs.eg.db (>= 3.19.1), rSpectral, WGCNA, ggplot2, ggrepel, minpack.lm, cowplot, data.table, scales, stats, Matrix

RdMacros Rdpack

Suggests knitr, BiocStyle, magick, rmarkdown, igraphdata, testthat (>= 3.0.0), vdiffr, devtools, pander, plotly, randomcoloR

Config/testthat/edition 3

VignetteBuilder knitr

BugReports https://github.com/lptolik/BioNAR/issues/

biocViews Software, GraphAndNetwork, Network

LazyData true

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

git_branch RELEASE_3_23

git_last_commit 19b17fe

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Colin Mclean [aut],
Anatoly Sorokin [aut, cre],
Oksana Sorokina [aut],
J. Douglas Armstrong [aut, fnd],
T. Ian Simpson [ctb, fnd]

Maintainer Anatoly Sorokin <lptolik@gmail.com>

Contents

addEdgeAtts . . . 4
annotateGeneNames . . . 4
annotateGoBP . . . 5
annotateGoCC . . . 6
annotateGoMF . . . 7
annotateGOont . . . 8
annotateInterpro . . . 9
annotatePresynaptic . . . 9
annotateSCHanno . . . 10
annotateTopOntoOVG . . . 11
annotateVertex . . . 12
applpMatrixToGraph . . . 13
BioNAR . . . 14
buildConsensusMatrix . . . 15
buildNetwork . . . 15
calcAllClustering . . . 16
calcBridgeness . . . 17
calcCentrality . . . 18
calcCentralityExternalDistances . . . 19
calcCentralityInternalDistances . . . 20
calcClustering . . . 21
calcDiseasePairs . . . 22
calcEntropy . . . 23
calcMembership . . . 24
calcReclusterMatrix . . . 25
calcSparsness . . . 26
clusteringSummary . . . 27
clusterORA . . . 28
compMembership . . . 29
degreeBinnedGDAs . . . 30
diseasome . . . 31
escapeAnnotation . . . 32
evalCentralitySignificance . . . 33
findLCC . . . 34
fitDegree . . . 34
fitSigmoid . . . 35
flatfile.go.BP.csv . . . 36

flatfile.go.CC.csv . . . 36
flatfile.go.MF.csv . . . 37
flatfile_human_gene2HDO.csv . . . 37
getAnnotationList . . . 37
getAnnotationVertexList . . . 38
getBridgeness . . . 39
getCentralityMatrix . . . 40
getClustering . . . 41
getClusterSubgraphByID . . . 42
getCommunityGraph . . . 43
getDiseases . . . 44
getDType . . . 44
getDYNAMO . . . 45
getEntropy . . . 46
getEntropyRate . . . 47
getGNP . . . 48
getGraphCentralityECDF . . . 49
getIDs . . . 49
getPA . . . 50
getRandomGraphCentrality . . . 51
getRobustness . . . 52
gofs . . . 53
law-class . . . 54
layoutByCluster . . . 54
layoutByRecluster . . . 55
makeConsensusMatrix . . . 55
makeMembership . . . 57
markBowTie . . . 58
metlMatrix . . . 58
normModularity . . . 59
permute . . . 61
plotBridgeness . . . 61
plotEntropy . . . 63
plotRatio . . . 64
plotSigmoid . . . 65
PPI_Presynaptic.csv . . . 65
PPI_Presynaptic.gml . . . 66
prepareGDA . . . 66
PresynAn.csv . . . 67
recluster . . . 67
removeVertexTerm . . . 68
runPermDisease . . . 69
sampleDegBinnedGDA . . . 70
sampleGraphClust . . . 71
SCH_flatfile.csv . . . 72
summaryStats . . . 72
unescapeAnnotation . . . 73
zeroNA . . . 74

Index **75**

---

addEdgeAtts *Copy edge attributes from one graph to another*

Description

Copy edge attributes from one graph to another

Usage

```
addEdgeAtts(GG, gg)
```

Arguments

GG	igraph object, source of attributes
gg	igraph object, attributes recipient

Value

annotated version of gg igraph object

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
GG <- igraph::read_graph(file, format="gml")
gg<-findLCC(GG)
gg <- addEdgeAtts(GG, gg)
edge_attr_names(gg)
```

---

annotateGeneNames *Annotate Human Gene Names*

Description

For the protein-protein interaction (PPI) or disease gene interaction (DGN) graphs that have EntrezID as a vertex name this function extract standard name from org.Hs.eg.db and annotate vertices.

Usage

```
annotateGeneNames(gg, orgDB = org.Hs.eg.db, keytype = "ENTREZID")
```

Arguments

gg	igraph object to annotate
orgDB	ordDB object, by default human is assumed from org.Hs.eg.db
keytype	type of IDs stored in the name vertex attribute, by default ENTREZID is assumed.

Details

If vertex name attrubite stores not EntrezID or network is build not from human genes, other OrgDb-class object could be provided in orgDB and one of keytypes from that object that correspond to the nature of the vertex name attrubite could be provided in the keytype attribute.

If for some vertices name attrubite does not match keys with particular keytypes in the orgDB object, empty string is added as GeneName.

Value

igraph object with new vertex attribute GeneName

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
paste(V(agg)$GeneName[idx], 'GRPEL1')
```

---

annotateGoBP *Add GO BP annotation to the graph vertices*

Description

The function loads an annotation data matrix called annoF, which contains three columns; the first containing gene Entrez IDs, the second gene GO BP ID terms, the third gene GO BP description terms. The function then performs a many-to-one mapping of each matrix row to a network vertex using matching Entrez IDs, filling the vertices attributes GO_BP_ID and GO_BP.

Usage

```
annotateGoBP(gg, annoF, idatt = "name")
```

Arguments

gg	graph to update
annoF	annotation matrix in Pair form
idatt	optional name of the vertex attribute to map to the annotation data.frame first column

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
sfile<-system.file("extdata", "flatfile.go.BP.csv", package = "BioNAR")
goBP <- read.table(sfile, sep="\t", skip=1, header=FALSE,
strip.white=TRUE, quote="")
sgg <- annotateGoBP(gg, goBP)
```

---

annotateGoCC *Add GO CC annotation to the graph vertices*

Description

The function loads an annotation data matrix called annoF, which contains three columns; the first containing gene Entrez IDs, the second gene GO ID terms, the third gene GO CC description terms. The function then performs a many-to-one mapping of each matrix row to a network vertex using matching Entrez IDs, filling the vertices attributes GO_CC_ID and GO_CC.

Usage

```
annotateGoCC(gg, annoF, idatt = "name")
```

Arguments

gg	graph to update
annoF	annotation matrix in Pair form
idatt	optional name of the vertex attribute to map to the annotation data.frame first column

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
sfile<-system.file("extdata", "flatfile.go.CC.csv", package = "BioNAR")
goCC <- read.table(sfile, sep="\t", skip=1, header=FALSE,
strip.white=TRUE, quote="")
sgg <- annotateGoCC(gg, goCC)
```

annotateGoMF *Add GO MF annotation to the graph vertices*

Description

The function loads an annotation data matrix called annoF, which contains three columns; the first containing gene Entrez IDs, the second gene GO MF ID terms, the third gene GO MF description terms. The function then performs a many-to-one mapping of each matrix row to a network vertex using matching Entrez IDs, filling the vertices attributes GO_MF_ID and GO_MF.

Usage

```
annotateGoMF(gg, annoF, idatt = "name")
```

Arguments

gg	graph to update
annoF	annotation matrix in Pair form
idatt	optional name of the vertex attribute to map to the annotation data.frame first column

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
sfile<-system.file("extdata", "flatfile.go.MF.csv", package = "BioNAR")
goMF <- read.table(sfile, sep="\t", skip=1, header=FALSE,
strip.white=TRUE, quote="")
sgg <- annotateGoMF(gg, goMF)
```

annotateGOont *Annotate nodes with GO terms*

Description

For the protein-protein interaction (PPI) or disease gene interaction (DGN) graphs that have EntrezID as a vertex name this function extract GeneOntolgy annotation from orgDB, which should be OrgDb-class, split them into three ontology group (MF,BP,CC) and annotate vertices with .

Usage

```
annotateGOont(gg, orgDB = org.Hs.eg.db, keytype = "ENTREZID", idatt = "name")
```

Arguments

gg	igraph object to annotate
orgDB	ordDB object, by default human is assumed from org.Hs.eg.db
keytype	type of IDs stored in the name vertex attribute, by default ENTREZID is assumed.
idatt	optional name of the vertex attributes that contains IDs matching the keytype

Details

If vertex name attrubite stores not EntrezID or network is build not from human genes, other OrgDb-class object could be provided in orgDB and one of keytypes from that object that correspond to the nature of the vertex name attrubite could be provided in the keytype attribute.

If for some vertices name attrubite does not match keys with particular keytypes in the orgDB object, empty string is added as GeneName.

Value

igraph object with new vertex attribute GeneName

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
ggGO <- annotateGOont(gg)
```

annotateInterpro *Add InterPro Family and Domain annotation to the graph vertices*

Description

Function takes data from annoF matrix and add them to attributes InterPro_Family for term and InterPro_Family_ID for IDs.

Usage

```
annotateInterpro(gg, annoF, annoD, idatt = "name")
```

Arguments

gg	graph to update
annoF	family annotation matrix in Pair form
annoD	domain annotation matrix in Pair form
idatt	optional name of the vertex attributes that contains Entrez IDs

Details

Function takes data from annoD matrix and add them to attributes InterPro_Domain for term and InterPro_Domain_ID for IDs.

Value

annotated igraph object

See Also

getAnnotationVertexList

annotatePresynaptic *Add presynaptic functional groups*

Description

Function takes from anno matrix manually curated presynaptic genes functional annotation derived from Boyken at al. (2013) doi:10.1016/j.neuron.2013.02.027 and add them to attributes PRESYNAPTIC.

Usage

```
annotatePresynaptic(gg, anno, idatt = "name")
```

Arguments

gg	graph to update
anno	annotation matrix in Pair form
idatt	optional name of the vertex attributes that contains Entrez IDs

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
sfile<-system.file("extdata", "PresynAn.csv", package = "BioNAR")
pres <- read.csv(sfile,skip=1,header=FALSE,strip.white=TRUE,quote="")
gg <- annotatePresynaptic(gg, pres)
```

---

annotateSCHanno *Add SCHanno synaptic functional groups*

Description

The function loads an annotation data matrix of functional groups for schizopherina risk genes (1) called anno, which contains three columns; the first containing gene Entrez IDs, the second gene functional group ID terms, the third gene functional group description terms. The function then performs a many-to-one mapping of each matrix row to a network vertex using matching Entrez IDs, filling the SCHanno vertices attribute.

Usage

```
annotateSCHanno(gg, anno, idatt = "name")
```

Arguments

gg	igraph object to annotate
anno	annotation matrix in Pairs form
idatt	optional name of the vertex attributes that contains Entrez IDs

Details

References:

1. Lips E, Cornelisse L, Toonen R, Min J, Hultman C, the International Schizophernia Consortium, Holmans P, Donovan M, Purcell S, Smit A, Verhage M, Sullivan P, Visscher P, D P: Functional gene group analysis identifies synaptic gene groups as risk factor for schizophrenia. Molecular Psychiatry 2012,17:996–1006.

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
afile<-system.file("extdata", "SCH_flatfile.csv", package = "BioNAR")
dis    <- read.table(afile, sep="\t", skip=1, header=FALSE,
strip.white=TRUE, quote="")
agg<-annotateSCHanno(gg, dis)
```

---

annotateTopOntoOVG *Annotate graph with disease terms*

Description

The function loads a human disease annotation matrix called `dis`, which contains three columns; the first containing gene Entrez IDs, the second gene Human Disease Ontology (HDO) ID terms, the third gene HDO description terms. For human protein-protein interaction (PPI) or disease-gene networks (DGN) that have human Entrez IDs for the igraph vertex name attribute. The function then performs a many-to-one mapping of each matrix row to a network vertex using matching Entrez IDs, filling the vertices attributes `TopOnto_OVG_HDO_ID` and `TopOnto_OVG`.

Usage

```
annotateTopOntoOVG(gg, dis, idatt = "name")
```

Arguments

`gg`	igraph object to annotate
`dis`	annotation matrix in Pairs form
`idatt`	optional name of the vertex attributes that contains Entrez IDs

Value

annotated igraph object

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
# read HDO data extracted from hxin/topOnto.HDO.db for synaptic network
afile<-system.file("extdata", "flatfile_human_gene2HDO.csv",
package = "BioNAR")
dis    <- read.table(afile, sep="\t", skip=1, header=FALSE,
strip.white=TRUE, quote="")
agg<-annotateTopOntoOVG(gg, dis)
```

---

annotateVertex *Generic annotation function*

Description

Function to build and fill a vertex attribute given an igraph object. Where parameter 'name' is the new vertex attribute name and values are filled from a two column data.frame supplied to 'value' attribute. The first containing vertex name IDs, and the second the vertex annotation value.

Usage

```
annotateVertex(gg, name, values, idatt = "name")
```

Arguments

gg	igraph object to annotate
name	name of the attribute
values	annotation data.frame
idatt	optional name of the vertex attribute to map to the annotation data.frame first column

Details

As a first step all attributes with provided names will be removed.

Value

igraph object where vertex attribute name contains annotation terms separated by semicolon.

See Also

getAnnotationVertexList

Examples

```
g1 <- make_star(10, mode="undirected")
V(g1)$name <- letters[1:10]
m<-rbind(data.frame(ID=letters[1:10], terms=letters[1:10]),
data.frame(ID=letters[1:10], terms=LETTERS[1:10]))
g2<-annotateVertex(g1, name='cap', values=m)
V(g2)$cap
```

`applpMatrixToGraph` *Add attributes to the vertex.*

Description

This function suits more for updating calculated vertex properties rather than node annotation. For the later case use `annotateVertex`.

Usage

```
applpMatrixToGraph(gg, m)
```

Arguments

`gg`	igraph object
`m`	matrix of values to be applied as vertex attributes. matrix should contains column "ID" to map value to the vertex.

Details

Unlike `annotateVertex`, which is able to collapse multiple annotation terms, this function assume that vertex ID values are unique in the `m` matrix and corresponds to the `name` vertex attribute. If graph has no `name` vertex attribute error will be raised.

Value

modified igraph object

See Also

annotateVertex

Examples

```
g1 <- make_star(10, mode="undirected")
V(g1)$name <- letters[1:10]
m<-cbind(ID=letters[1:10],capital=LETTERS[1:10])
g1<-BioNAR::applpMatrixToGraph(g1,m)
V(g1)$capital
```

BioNAR *BioNAR: Biological Network Analysis in R*

Description

The R package BioNAR, developed to step by step analysis of PPI network. The aim is to quantify and rank each protein's simultaneous impact into multiple complexes based on network topology and clustering. Package also enables estimating of co-occurrence of diseases across the network and specific clusters pointing towards shared/common mechanisms.

Author(s)

Maintainer: Anatoly Sorokin <lptolik@gmail.com>

Authors:

- Colin Mclean <Colin.D.Mclean@ed.ac.uk>
- Oksana Sorokina <oksana.sorokina@ed.ac.uk>
- J. Douglas Armstrong <Douglas.Armstrong@ed.ac.uk> [funder]

Other contributors:

- T. Ian Simpson <Ian.Simpson@ed.ac.uk> [contributor, funder]

See Also

Useful links:

- Report bugs at https://github.com/lptolik/BioNAR/issues/

`buildConsensusMatrix` *Build a consensus matrix from list of resampled clustering matrices outputted from the function* `sampleGraphClust`

Description

Build a consensus matrix from list of resampled clustering matrices outputted from the function `sampleGraphClust`

Usage

```
buildConsensusMatrix(lcc)
```

Arguments

`lcc`	list of membership matrices obtained from the `sampleGraphClust`

Details

Function build a consensus matrix from list of membership matrices, which are a three column matrix: the first column contains the vertex IDs of input network; the second column the vertex IDs of the subsampled network, or -1 if the vertex has been masked; the third column the cluster membership of subsampled network, or -1 if vertex has been masked. The randomised resampled membership matrices could be obtained from the function `sampleGraphClust`.

Value

consensus matrix of Nvert X Nvert

`buildNetwork` *Build network from data.table*

Description

Wrapper for `graph_from_data_frame` function which will always return the largest connect component for a given network `ff`. The function will also annotated the edges in `ff` with PubMed data from `kw` if provided.

Usage

```
buildNetwork(ff, kw = NA, LCC = TRUE, simplify = TRUE)
```

Arguments

ff	network structure data.frame with first two columns defining the network edge nodes
kw	pmid keyword annotation data.frame. If NA no annotation will be added
LCC	if TRUE only largest connected component is returned
simplify	if TRUE loops and multiple edges will be removed

Value

igraph object of the largest connected component

Examples

```
f<-data.frame(A=c('A', 'A', 'B', 'D'), B=c('B', 'C', 'C', 'E'))
gg<-buildNetwork(f)
V(gg)$name
```

---

calcAllClustering *Calculate memberships for all clustering algorithms and store them on the graph vertices.*

Description

This function will call calcClustering for each clustering algorithm given in our predefined list. In the event no clustering could be performed, warnings will be issued and no new vertex attribute added to the graph.

Usage

```
calcAllClustering(gg, weights = NULL)
```

Arguments

gg	graph for analysis
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.

Value

new graph object with all membership results stored as a vertex attribute.

See Also

calcClustering

Examples

```
g1 <- make_star(10, mode="undirected")
V(g1)$name <- letters[1:10]
g1<-calcAllClustering(g1)
clusteringSummary(g1)
```

`calcBridgeness` *Helper function that uses* `getBridgeness` *to calculate graph node bridgeness values for selected algorithm and consensus matrix and save them as a graph attribute* `BRIDGENESS.<alg>` *with* `<alg>` *replaced by the selected algorithm name.*

Description

Helper function that uses `getBridgeness` to calculate graph node bridgeness values for selected algorithm and consensus matrix and save them as a graph attribute `BRIDGENESS.<alg>` with `<alg>` replaced by the selected algorithm name.

Usage

```
calcBridgeness(gg, alg, conmat)
```

Arguments

`gg`	igraph object
`alg`	clustering algorithm
`conmat`	consensus matrix calculated with that algorithm

Value

graph with additional attributes to store Bridgeness value

See Also

getBridgeness

Examples

```
library(BioNAR)
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
set.seed(100)
gg <- calcClustering(karate, 'louvain')
cnmat <- makeConsensusMatrix(gg, N=10, alg = 'louvain', type = 2, mask = 10)
gg<-calcBridgeness(gg, alg = 'louvain', cnmat)
hist(V(gg)$BRIDGENESS.louvain)
```

---

calcCentrality *Calculate the vertex centrality measures*

Description

Calculate the vertex centrality measures (degree, betweenness, closeness, semi-local, etc....) for each graph vertex and store each result as new vertex attribute in the graph.

Usage

```
calcCentrality(gg, weights = NULL)
```

Arguments

gg	igraph object
weights	Possibly a numeric vector giving edge weights. If this is NULL and the graph has a weight edge attribute, then the attribute is used. If this is NA then no weights are used (even if the graph has a weight attribute).

Details

A wrapper function that first calls `getCentralityMatrix`, to calculate all vertex centrality measures, and then `applpMatrixToGraph` to store each centrality result as a new vertex attribute in the graph. The use of `weights` explained in details in `getCentralityMatrix`.

Value

modified igraph object

See Also

`getCentralityMatrix()`

Examples

```
data(karate,package='igraphdata')
ggm<-calcCentrality(karate)
V(ggm)$DEG
```

calcCentralityExternalDistances *Function to calculate a distance matrix between a list of permuted vertex centrality matrices and a unperturbed reference matrix.*

Description

Function to calculate a distance matrix between a list of permuted vertex centrality matrices and a unperturbed reference matrix.

Usage

```
calcCentralityExternalDistances(m, l, keepOrder = FALSE, dist = "euclidean")
```

Arguments

m	reference matrix, for example centrality obtained by invocation getCentralityMatrix
l	list of permuted matrix, for example centrality obtained by invocation getRandomGraphCentrality
keepOrder	if FALSE valuess will be sorted
dist	methods available from dist function

Value

matrix with seven columns containing distances between each element of l and reference matrix m

See Also

getRandomGraphCentrality

getCentralityMatrix

calcCentralityInternalDistances

Examples

```
data(karate,package='igraphdata')
m<-getCentralityMatrix(karate)
gnp<-list()
for(i in 1:10){
    gnp[[i]]<-getRandomGraphCentrality(karate,type = 'gnp')
}
gnpEDist<-calcCentralityExternalDistances(m,gnp)
summary(gnpEDist)
```

`calcCentralityInternalDistances` *Function calculates a set of distance metrics between each vertex pair given a list of vertex centrality matrices*

Description

Function calculates a set of distance metrics between each vertex pair given a list of vertex centrality matrices

Usage

```
calcCentralityInternalDistances(l, keepOrder = FALSE, dist = "euclidean")
```

Arguments

`l`	list of matrices, for example centrality obtained by invocation `getRandomGraphCentrality`
`keepOrder`	if FALSE values will be sorted before distance calculations
`dist`	methods available from `dist` function

Value

matrix with seven columns containing distances between all pairs of `l` elements.

See Also

getRandomGraphCentrality

getCentralityMatrix

calcCentralityExternalDistances

Examples

```
data(karate,package='igraphdata')
m<-getCentralityMatrix(karate)
gnp<-list()
for(i in 1:10){
    gnp[[i]]<-getRandomGraphCentrality(karate,type = 'gnp')
}
gnpIDist<-calcCentralityInternalDistances(gnp)
summary(gnpIDist)
```

`calcClustering` *Calculate community membership for given clustering algorithm and store the results as new vertex attributes in the graph..*

Description

When applying resampling the clustering results of a clustering algorithm applied to a graph can differ due to the stochastic nature of the resampling algorithm. To allow reproducible downstream analysis clustering results are stored as vertex attributes in the graph. This function call `getClustering` and stores community membership as new vertex attribute in the graph, and Modularity as a new graph attribute prefix with the `alg` name.

Usage

```
calcClustering(gg, alg, weights = NULL)
```

Arguments

`gg`	igraph object to cluster
`alg`	algorithm to apply
`weights`	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the `spectral` clustering.

Details

NOTE: `getClustering` verifies algorithm names with `match.arg` so correct membership will be calculated, but name of the attribute is taken from `alg` argument, so it is possible that vertex attribute name won't exactly match name of the algorithm from `link{getClustering}`.

Value

modified igraph object with calculated membership stored as a vertex attribute and modularity as a graph attribute

See Also

getClustering

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
g<-calcClustering(karate, 'louvain')
vertex_attr_names(g)
graph_attr(g, 'louvain')
```

`calcDiseasePairs` *Calculate each disease-disease pair overlap given a list of disease terms.*

Description

Calculate each disease-disease pair overlap (or separation) on a given PPI network model, based on analysis described in Menche et al. 2015

Usage

```
calcDiseasePairs(
  gg,
  name,
  diseases = NULL,
  permute = c("none", "random", "binned")
)
```

Arguments

`gg`	interactome network as igraph object
`name`	name of the attribute that stores disease annotation
`diseases`	list of diseases to match
`permute`	type of permutations. `none` – no permutation is applied, `random` – annotation is randomly shuffled, `binned` – annotation is shuffled in a way to preserve node degree-annotation relationship by `degreeBinnedGDAs`.

Value

list with three matrices:

- disease_separation – Ndisease X Ndisease matrix of separations
- gene_disease_separation – Ngenes X Ndisease+2 matrix of gene-disease separation
- disease_localisation – matrix with diseases in rows and number of genes (N), average and standard deviation of gene-disease separation in columns

References

Menche, J. et al. Uncovering disease-disease relationships through the incomplete interactome. Science, 347, (6224):1257601 (2015).

See Also

degreeBinnedGDAs

sampleDegBinnedGDA

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
paste(V(agg)$GeneName[idx], 'GRPEL1')
p <- calcDiseasePairs(
agg,
name = "TopOntoOVGHDOID",
diseases = c("DOID:10652", "DOID:3312", "DOID:12849"),
permute = "n"
)
p$disease_separation
```

`calcEntropy` *Calculate the graph entropy for each perturbed vertex, and save the results as new vertex attributes in the graph.*

Description

This function calculate the graph entropy for each perturbed vertex by calling `getEntropy`, and save the results as new vertex attributes SR_UP and SR_DOWN in the graph.

Usage

```
calcEntropy(gg, maxSr = NULL, exVal = NULL)
```

Arguments

`gg`	igraph object
`maxSr`	the maximum entropy rate $maxSR$, if NULL `getEntropyRate` will be called.
`exVal`	expression values boundaries. Two columns are expected: `xx` and `lambda`. If NULL default values `c(2,14)` and `c(-14,14)` will be used for `xx` and `lambda` respectively.

Details

According to Teschendorf et al., 2010, network entropy measure quantifies the degree of randomness in the local pattern information flux around single genes. For instance, in metastatic cancer this measure was found significantly higher than in non-metastatic and helped to identify genes and entire pathways involved on metastasis. However, for the assessment of scale-free structure we do not actually require gene expression data as it based solely on the network topology.

Value

graph with SR_UP and SR_DOWN vertex attributes storing the graph entropy values with over- or under-expressing each vertex.

See Also

`getEntropy()`

Other Entropy Functions: `getEntropy()`, `getEntropyRate()`, `plotEntropy()`

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
gg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(gg)$name == '80273')
paste(V(gg)$GeneName[idx], 'GRPEL1')
gg<- calcEntropy(gg)
```

---

calcMembership *Calculate cluster memberships for the graph.*

Description

Calculates the clustering membership for one of the 10 clustering algorithms defined in function `getClustering`

Usage

```
calcMembership(
  gg,
 alg = c("lec", "wt", "fc", "infomap", "louvain", "sgG1", "sgG2", "sgG5", "spectral"),
  weights = NULL
)
```

Arguments

`gg`	igraph object to cluster
`alg`	algorithm name
`weights`	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If it is NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph has a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the `spectral` clustering.

Value

data.frame with columns `names` and `membership`

See Also

getClustering

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
m<-calcMembership(karate, 'lec')
head(m)
```

`calcReclusterMatrix` *Hierarchical graph clustering*

Description

This function takes in a `gg` and initial vertex community membership values `mem` as returned by `calcMembership`, and then performs a reclustering of the graph given the clustering algorithm `alg` to those clusters of size greater than `CnMAX`

Usage

```
calcReclusterMatrix(
  gg,
  mem,
  alg,
  CnMAX = 10,
  weights = NULL,
  keepSplit = FALSE
)
```

Arguments

`gg`	graph to cluster
`mem`	data.frame with previous level clustering results
`alg`	algorithm to apply
`CnMAX`	maximus size of the cluster in `mem` that will not be processed
`weights`	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the `spectral` clustering.

keepSplit logical, wether to keep previous membership in the output matrix

Value

remembership matrix, that contains vertex ID membership and result of reclustering

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
remem<-calcReclusterMatrix(karate,mem,alg,10)
```

---

calcSparsness *Calculate sparsness of the graph.*

Description

For a simple unweighted, undirected graph G(N,E). Network sparseness is defined as the ratio of the actual number of graph edges (E) to the maximum number of edges possible in a graph with same number of vertices (N): E/binom(N,2)

Usage

```
calcSparsness(gg)
```

Arguments

gg graph to evaluate

Value

sparsness value

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
calcSparsness(gg)
```

clusteringSummary *Matrix of cluster characteristics*

Description

Function to calculate basic summary statistics after apply clustering algorithm:

- N – number of vertices in the graph vcount
- mod – clustering modularity modularity, the ratio of edges found within communities to the number of edges found between communities, relative to a randomised model
- C – number of clusters
- Cn1 – number of singletones (clusters of size 1)
- Cn100 – number of clusters containing more than 100 nodes
- mu – the ratio of edges found within communities to the number of edges found between communities
- Min. C – minimum of the cluster size
- 1st Qu. C – first quartile of the cluster size
- Median C – median of the cluster size
- Mean C – average cluster size
- 3rd Qu. C – third quartile of the cluster size
- Max. C – maximum of the cluster size

Usage

```
clusteringSummary(
  gg,
 att = c("lec", "wt", "fc", "infomap", "louvain", "sgG1", "sgG2", "sgG5", "spectral")
)
```

Arguments

gg	graph to analyse
att	vector of attribute names that contains membership data

Value

matrix of clustering characteristics

Examples

```
data(karate,package='igraphdata')
g<-calcAllClustering(karate)
clusteringSummary(g)
```

`clusterORA` *Calculate annotation enrichment for clusters in the graph*

Description

Calculate the cluster enrichment of a graph given a clustering algorithm `alg` and vertex annotation attribute 'name'. Function generates an enrichment table, one row for each cluster, containing: size of the cluster (`Cn`), number of annotated vertices in the graph F_n (`Fn`), number of annotated vertices in the cluster μ (`Mu`), odds ratio (`OR`) and its 95% Confidence interval $[CI_l, CI_u]$ (`CIl` and `CIu`), two fold enrichment values F_e (`Fe`) and F_c (`Fc`). We also provide the list of vertices from the cluster that contribute to the annotation term, p.value of enrichment (`pval`) and depletion (`palt`) using the Hypergeometric test, adjusted p.values using Benjamini and Yekutieli correction (BY).

Usage

```
clusterORA(g, alg, name, vid = "name", alpha = 1, col = COLLAPSE)
```

Arguments

`g`	graph to get annotation from
`alg`	cluster algorithm and membership attribute name
`name`	annotation attribute name
`vid`	attribute to be used as a vertex ID
`alpha`	probability threshold
`col`	list separation character in attribute, by default is ;

Details

Given the enrichment results, we can calculate the log of the Odds Ratio (`OR`) as:

$$\ln(OR) = \ln(\frac{\mu(N - F_n + \mu - C_n)}{(C_n - \mu)(F_n - \mu)})$$

and it's upper and lower 95% Confidence Interval:

$$CI(\ln(OR)) = \ln(OR) \pm 1.96\sqrt{\frac{1}{\mu} + \frac{1}{C_n - \mu} + \frac{1}{F_n - \mu} + \frac{1}{N - F_n + \mu - C_n}}$$

Using the odds ratio allows us to distinguish functionally enriched communities relative to functionally depleted communities.

Two types of fold enrichment values calculated as follow:

$$F_e = \frac{(\frac{\mu}{F_n})}{(\frac{C_n}{N})}$$

$$F_c = \frac{(\frac{\mu}{C_n})}{(\frac{C_n}{N})}$$

Value

A table with overrepresentation results. Each row corresponds to a tested annotation in particular cluster. The columns are the following:

- alg – name of the clustering algorithm;
- cl – cluster ID;
- FL – name of the enriched term;
- N – number vertices in the network;
- Fn – number of vertices in the graph annotated by term `Fl` (F_n);
- Cn – size of the cluster;
- Mu – number of vertices in the cluster annotated by term `Fl` (μ);
- OR – odds ratio ;
- CIl – odds ratio 95% confidence interval lower bound (CI_l);
- CIu – odds ratio 95% confidence interval upper bound(CI_u);
- Fe – fold enrichment F_e;
- Fc – fold enrichment F_c;
- pval – an enrichment p-value from hypergeometric test;
- padj – a BY-adjusted p-value;
- palt – an depletion p-value from hypergeometric test;
- paltadj – a BY-adjusted depletion p-value;
- overlapGenes – vector with overlapping genes.

Examples

```
options("show.error.messages"=TRUE)
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
g <- igraph::read_graph(file, format="gml")
anL<-getAnnotationVertexList(g, 'TopOntoOVGHDOID')
res<-clusterORA(g, alg='louvain', name='TopOntoOVGHDOID', vid='name')
andf<-unique(data.frame(ID=vertex_attr(g, 'TopOntoOVGHDOID'),
Term=vertex_attr(g, 'TopOntoOVG')))
rr<-merge(andf, res, by.y='FL', by.x='ID')
rr[order(rr$cl), ]
```

---

compMembership *Calculate cluster memberships for one of the graph component.*

Description

Calculates the clustering membership for one of the 10 clustering algorithms defined in function `getClustering` for selected graph component

Usage

```
compMembership(
  gg,
 alg = c("lec", "wt", "fc", "infomap", "louvain", "sgG1", "sgG2", "sgG5", "spectral"),
  compnum = 0,
  weights = NULL
)
```

Arguments

gg	igraph object to cluster
alg	algorithm name
compnum	number of the componet to cluster
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If it is NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph has a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.

Value

data.frame with columns names and membership

See Also

getClustering

---

degreeBinnedGDAs *Prepare mapping for degree-aware annotation shuffling.*

Description

Function to randomly shuffle vertex annotation terms, whilst preserving the vertex degree originally found with that annotation term.

Usage

```
degreeBinnedGDAs(gg, GDA, dtype)
```

Arguments

gg	graph to analyse
GDA	vertex annotations returned by prepareGDA
dtype	list of unique annotation terms to analyze

Value

mapping matrix between vertices, vertex-degree groups and annotation terms.

See Also

prepareGDA

getAnnotationList

sampleDegBinnedGDA

Examples

```
options("show.error.messages"=TRUE)
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
paste(V(agg)$GeneName[idx], 'GRPEL1')
gda<-prepareGDA(agg, 'TopOntoOVGHDOID')
m<-degreeBinnedGDAs(agg, gda, getAnnotationList(gda))
c(dim(m), vcount(agg), length(getAnnotationList(gda)))
head(m)
```

---

diseasome *Barabasi's Diseasome Network*

Description

In the paper Goh.t al. (2007) doi:10.1073/pnas.0701361104 Barabasi with colleagues published Diseasome: a network of disorders and disease genes linked by known disorder–gene associations. We extract definition of the genes, disorders and interactions from papers supplementary materials and store it as `graph` object.

Usage

```
diseasome
```

Format

A bipartite graph as `graph` object.

Vertex attributes: 'name' for the node ID, 'Name' for the human readable node name, 'Disorder.class', 'Type' for the human readable node type, 'label' and 'shape' for plotting the graph, 'type' the node type for bipartite `graph` representation.

Details

Diseasesome is a bipartite graph that have nodes of two types `gene` and `disease` and links are allowed only between nodes of different types. It could be projected to Human Disease Network (HDN) and Disease Gene Network (DGN).

Source

Goh, K.-I. et al. The human disease network. Proc. Natl. Acad. Sci. U.S.A. 104, 8685–8690 (2007). https://pnas.org/doi/full/10.1073/pnas.0701361104

`escapeAnnotation` *Escapes elements of list in annotation.*

Description

In situations when a given list of annotation ID terms may not be well formatted, and therefore not be interoperated as unique. For example, given a list of HDO IDs: HDO:14, HDO:143, HDO:1433, and HDO:14330, a grep for the term HDO:14 could return: HDO:143, HDO:1433, HDO:14330. To avoid this all terms should be enclosed in escape characters, which unlikely to find within annotation itself.

Usage

```
escapeAnnotation(annVec, col = COLLAPSE, esc = ESC)
```

Arguments

`annVec`	vector of annotation strings
`col`	term list separator character
`esc`	escape character

Details

NOTE: spaces are treated as regular characters, no trimming is applied before or after escaping.

Value

vector of annotation strings with elements escaped

See Also

unescapeAnnotation

Examples

```
annVec<-apply(matrix(letters, ncol=13), 2, paste, collapse=';')
cbind(annVec, escapeAnnotation(annVec, ';', '|'))
```

`evalCentralitySignificance` *Compare distance distributions of internal and external distances*

Description

Function to compare two distance distributions using the Kolmogorov-Smirnov test. Where the first distance distribution is generated internally and calculates the distance between random graph centralities. The second distance distribution is generated externally, and measures the distance between random and the original graph centralities.

Usage

```
evalCentralitySignificance(dmi, dme)
```

Arguments

`dmi`	distribution of internal distances between random graph centralities
`dme`	distribution of external distances between random and original graph centralities

Value

list of lists for each centrality value in the input matrix three element list is created where `ks` contains Kolmogorov-Smirnov test result from class `ks.test`; `pval` contains Kolmogorov-Smirnov test pvalue; and `dt` contains input distribution.

See Also

ks.test

Examples

```
data(karate,package='igraphdata')
m<-getCentralityMatrix(karate)
gnp<-list()
for(i in 1:10){
    gnp[[i]]<-getRandomGraphCentrality(karate,type = 'gnp')
}
gnpIDist<-calcCentralityInternalDistances(gnp)
gnpEDist<-calcCentralityExternalDistances(m,gnp)

simSig<-evalCentralitySignificance(gnpIDist,gnpEDist)
sapply(simSig,function(.x).x$ks$p.value)
```

findLCC *Find Largest Connected Component of the graph*

Description

Find Largest Connected Component of the graph

Usage

```
findLCC(GG)
```

Arguments

GG	igraph object to analyze

Value

igraph representation LCC

Examples

```
g1 <- make_star(10, mode="undirected") %du% make_ring(7) %du% make_ring(5)
lcc<-findLCC(g1)
summary(lcc)
```

fitDegree *Fit Power Law to degree distribution.*

Description

Fit a Powerlaw distribution to graph's degree distribution using the R "PoweRlaw" package (version 0.50.0) (Gillespie, 2015)

Usage

```
fitDegree(
  DEG,
  Nsim = 100,
  plot = FALSE,
  DATAleg = "Fit power-law",
  threads = 4,
  WIDTH = 480,
  HEIGHT = 480,
  legpos = "bottomleft",
  showErr = TRUE
)
```

Arguments

DEG	degree distribution
Nsim	number of bootstrap iterations
plot	logical, do you want plot to be drawn
DATAleg	legend string for degree data
threads	number of parallel computational threads
WIDTH	width of the plot in ptx
HEIGHT	heigth of the plot in ptx
legpos	position of the legend @seealso legend()
showErr	logical, do you want error on the plot legend

Value

an object of class law-class with results of fitting

Examples

```
##No: of bootstrap iterations use nsim > 100 for reliable result
nsim <- 10

##Legend Titles
Legend <- "Presynaptic PPI"

file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
pFit <- fitDegree( as.vector(igraph::degree(graph=gg)),
DATAleg=Legend,threads=1, Nsim=nsim)
```

---

fitSigmoid *Fit Fold-enrichment distribution to sigmoid function*

Description

This function calculates fit of the Fold-Enrichment distribution to the sigmoid function with the levels of noise specidied in SDV for all clustering algorithms, which have non-zero SUM3$`Psig&ORsig` in the enrichment table summary results. The function returns the list in which each element contains result for one of the noise level.

Usage

```
fitSigmoid(stat, SDv = c(0, 0.05, 0.1, 0.5))
```

Arguments

stat	enrichment results obtained from summaryStats
SDv	vector of noise SD values

Details

Results are repersented as a list with five elements:

- gridplot that allow comparison of fitting for different clustering algorithms;
- plots the list of individual plots from gridplot;
- fitInfo the data.frame that contains results of fitting, such as message, number of iterations and exit code;
- parInfo values and standard deviations for all sigmoid parameters;
- ks table of Kolmogorov-Smirnov test p-values.

Grid plot is designed in a way to be viewed in the device at least 12 inches in width and 12 inches in height.

Value

list of fitted functions tables and plots

See Also

summaryStats()

---

flatfile.go.BP.csv *Annotation from Gene Ontology Biological Process (GO_BP)*

Description

Annotation, downloaded from Gene Ontology for Biological Proceess domain. The table has columns: the first containing gene gene functional group ID terms, the second gene functional group description terms, the third - Human gene Entrez IDs; in csv format

See Also

annotateGoBP

---

flatfile.go.CC.csv *Annotation from Gene Ontology Cellular Compartment (GO_CC)*

Description

Annotation, downloaded from Gene Ontology for Cellular Compartment domain. The table has columns: the first containing gene gene functional group ID terms, the second gene functional group description terms, the third - Human gene Entrez IDs; in csv format

See Also

annotateGoCC

flatfile.go.MF.csv *Annotation from Gene Ontology Molecular Function (GO_MF)*

Description

Annotation, downloaded from Gene Ontology for Molecular Function domain. The table has columns: the first containing gene gene functional group ID terms, the second gene functional group description terms, the third - Human gene Entrez IDs; in csv format

See Also

annotateGoMF

flatfile_human_gene2HDO.csv *Human Gene Disease Associations (GDA)*

Description

Annotation derived from Human Disease Ontology database (HDO). The table contains three columns; the first containing gene Entrez IDs, the second gene Human Disease Ontology (HDO) ID terms, the third gene HDO description terms; in csv format

See Also

annotateTopOntoOVG

getAnnotationList *Extract unique values from annotations.*

Description

It is not uncommon to find both duplicated vertex annotation terms, and vertices annotated with multiple terms, in a given annotation list. This function creates a vector of unique annotation terms for each vertex given an input annotation list.

Usage

```
getAnnotationList(
  annVec,
  col = COLLAPSE,
  sort = c("none", "string", "frequency")
)
```

Arguments

annVec	vector of annotation strings
col	list separator character
sort	how to sort the result list

Value

vector of unique annotation terms

See Also

getAnnotationVertexList

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
annVec<-V(gg)$TopOntoOVG
al<-getAnnotationList(annVec)
al
```

---

getAnnotationVertexList

Return vertex list for each term in annotation attribute

Description

For different purposes annotation of graph vertices could be represented in three forms:

Pairs dataframe with vertex ID and annotation terms

Vertex Annotation list named with vertex ID and containing terms annotating each vertex

Annotation Vertices list named with term and containing vertex IDs

Usage

```
getAnnotationVertexList(g, name, vid = "name", col = COLLAPSE)
```

Arguments

g	graph to get annotation from
name	annotation attribute name
vid	attribute to be used as a vertex ID
col	list separation character in attribute, by default is ;

Details

This function takes Vertex Annotation from vertex attribute and convert it to Annotation Vertices form.

Value

named list with annotation in Annotation Vertices form

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
avl<-getAnnotationVertexList(gg, 'TopOntoOVGHDOID')
head(avl)
```

---

getBridgeness *Calculate bridginess from consensus matrix*

Description

Bridginess takes into account a vertices shared community membership together with its local neighbourhood. It was proposed in Nepusz et al., 2008 doi:10.1103/PhysRevE.77.016107.

Usage

```
getBridgeness(gg, alg, conmat)
```

Arguments

gg	igraph object
alg	clustering algorithm
conmat	consensus matrix calculated with that algorithm

Details

Function assumes clustering already been performed by the clustering algorithm, and its membership values stored in vertex attributes. If clustering algorithm vertex `alg` attribute is not found an error will be issued.

Value

data.frame with first column contains vertex ID, if GeneName attribute assigned to the vertices its value will be stored as a second column, the last column contains bridginess values for the

Examples

```
library(BioNAR)
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
gg <- calcClustering(karate, 'louvain')
cnmat <- makeConsensusMatrix(gg, N=10, alg = 'louvain', type = 2, mask = 10)
br<-getBridgeness(gg, alg = 'louvain', cnmat)
```

`getCentralityMatrix` *Calculate centrality measures for graph nodes.*

Description

Calculate centrality measures for graph nodes.

Usage

```
getCentralityMatrix(gg, weights = NULL)
```

Arguments

`gg`	igraph object
`weights`	Possibly a numeric vector giving edge weights. If this is NULL and the graph has a weight edge attribute, then the attribute is used. If this is NA then no weights are used (even if the graph has a weight attribute).

Details

The edge attribute `weights` treated differently by different functions calculating centrality measures. For example, `betweenness` use `weights` as an edge length, while in `page_rank` "an edge with a larger weight is more likely to be selected by the surfer", which infer the opposite meaning. Taking into account that all methods in `getClustering` treat edge `weights` in the same way as `page_rank`, we calculate the `distance=1/weights` as edge weights for `BET`, `dBET`, `mnSP`, and `sdSP` values. So we treat `weights` in the package consistently as the strength and closiness of vertices, rather the distance between them.

Value

data.frame with following columns:

- ID - vertex ID
- DEG - degree
- iDEG - in-degree (directed graph only)
- oDEG - out-degree (directed graph only)
- BET - betweenness for undirected graph

- dBET - betweenness when directionality is taken into account (directed graph only)
- CC - clustering coefficient
- SL - semilocal centrality
- mnSP - mean shortest path
- PR - page rank for undirected graph
- dPR - page rank when directionality is taken into account (directed graph only)
- sdSP - standard deviation of the shortest path

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
m<-getCentralityMatrix(gg)
```

`getClustering` *Get clustering results for the graph.*

Description

Wrapper function for calculation of clustering for predefined set of ten algorithms:

- lec – leading eigenvector community (version of `cluster_leading_eigen`), directed graph will be converted to undirected by `as_undirected` with mode `collapse`;
- wt – walktrap community `cluster_walktrap`;
- fc – fastgreedy community `cluster_fast_greedy`, directed graph will be converted to undirected by `as_undirected` with mode `collapse`;
- infomap – infomap community `cluster_infomap`;
- louvain – cluster_louvain `cluster_louvain`, directed graph will be converted to undirected by `as_undirected` with mode `collapse`;
- sgG1 – spin-glass model and simulated annealing clustering (version of `cluster_spinglass` with spins=500 and gamma=1);
- sgG2 – spin-glass model and simulated annealing clustering (version of `cluster_spinglass` with spins=500 and gamma=2);
- sgG5 – spin-glass model and simulated annealing clustering (version of `cluster_spinglass` with spins=500 and gamma=7);
- spectral – spectral modularity clustering `spectral_igraph_communities`;

Usage

```
getClustering(
  gg,
 alg = c("lec", "wt", "fc", "infomap", "louvain", "sgG1", "sgG2", "sgG5", "spectral"),
  weights = NULL
)
```

Arguments

gg	igraph object to cluster
alg	clustering algorithm name
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.

Details

graph suppose to be undirected. If algorithm failed warning will be issued and function returned NULL.

Algorithm names are verified with match.arg.

Value

communities object or NULL if algorithm failed.

Examples

```
data(karate,package='igraphdata')
c<-getClustering(karate,'lec')
c$modularity
```

---

getClusterSubgraphByID

Return induced subgraph for cluster

Description

Function reads in a graph gg, vertex cluster membership vector mem, and returns an induced subgraph given a cluster membership number 'clID'.

Usage

```
getClusterSubgraphByID(clID, gg, mem)
```

Arguments

clID	cluster ID to extracte
gg	graph to analyze
mem	membership vector

Value

induced subgraph as igraph object

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
c<-getClustering(karate,alg = alg)
gc3<-getClusterSubgraphByID(3,karate,membership(c))
#plot(gc3,vertex.label=V(gc3)$name)
```

`getCommunityGraph` *Create new graph with communities as a nodes.*

Description

The idea based upon this StackOverflow answer

Usage

```
getCommunityGraph(gg, membership)
```

Arguments

`gg`	graph to convert
`membership`	participation list for new graph

Value

community graph

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
cg<-getCommunityGraph(karate,mem$membership)
```

getDiseases *Get HDO disease IDs*

Description

Return vector of HDO disease IDs for synaptic PPI analysis.

Usage

```
getDiseases()
```

Value

vector of disease IDs of interest

See Also

getDType

Examples

```
getDiseases()
```

getDType *Get DiseaseTypes*

Description

Return vector of disease abbreviations for synaptic PPI analysis.

Usage

```
getDType()
```

Value

vector of disease abbreviations for synaptic PPI analysis.

See Also

getDiseases

Examples

```
getDType()
```

getDYNAMO *Calculate DYNAMO sensitivity matrix.*

Description

This function calculates sensitivity matrix that represents perturbation patterns defined by topology and edge weights of the network. If weights are signed value sensitivity matrix is able to reproduce not only activation but inhibition relationships in the network.

Usage

```
getDYNAMO(g, attr = NULL, vid = "name", alpha = 0.9)
```

Arguments

g	igraph object
attr	NULL or the name of edge attribute containing numerical weight values
vid	name of the vertex attribute to be used as row and column names
alpha	parameter characterizing the propagation strength, default value 0.9 taken from Santolini paper.

Details

Algorithm proposed in:

Santolini,M. and Barabasi,A.-L. (2018) Predicting perturbation patterns from the topology of biological networks. Proc Natl Acad Sci USA, 169, 201720589.

Value

sparce sensitivity matrix defined by the network topology and edge values

Examples

```
data(karate, package='igraphdata')
upgrade_graph(karate)
d<-getDYNAMO(karate,attr='weight')
df<-metlMatrix(d)
head(df)
```

`getEntropy` *Calculates vertex perturbation graph entropy.*

Description

According to Teschendorf et al., 2010, network entropy measure quantifies the degree of randomness in the local pattern information flux around single genes. For instance, in metastatic cancer this measure was found significantly higher than in non-metastatic and helped to identify genes and entire pathways involved on metastasis. However, for the assessment of scale-free structure we do not actually require gene expression data as it based solely on the network topology.

Usage

```
getEntropy(gg, maxSr = NULL, exVal = NULL)
```

Arguments

`gg`	igraph object
`maxSr`	the maximum entropy rate $maxSR$, if NULL `getEntropyRate` will be called.
`exVal`	expression values boundaries. Two columns are expected: `xx` and `lambda`. If NULL default values `c(2,14)` and `c(-14,14)` will be used for `xx` and `lambda` respectively.

Details

In this function, following procedure described in (Teschendorff et al., 2015), all vertexes are artificially assigned a uniform weight then sequentially perturbed with the global entropy rate (SR) after each protein's perturbation being calculated and plotted against the log of the protein's degree. In case of scale-free or approximate scale-free topologies, we see a clear bi-modal response between over-weighted vertices and their degree and an opposing bi-phasic response in under-weighted vertices and their degrees.

Value

matrix containing for each Gene:

- Entrez ID,
- Name,
- Degree,
- UP – Graph Entropy values when gene is expressed up,
- DOWN – Graph Entropy values when gene is expressed down.

Note

Entropy is calculated with respect to GeneName property, if there is no such vertex attribute in the graph vertex name will be copied to the GeneName attribute. If any NA is found in GeneNames error will be thrown.

See Also

Other Entropy Functions: `calcEntropy()`, `getEntropyRate()`, `plotEntropy()`

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
gg<-annotateGeneNames(gg)
any(is.na(V(gg)$GeneName))
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(gg)$name == '80273')
paste(V(gg)$GeneName[idx], 'GRPEL1')
e<- getEntropy(gg)
```

`getEntropyRate` *Calculate the maximum entropy rate and initial entropy rate .*

Description

This function calculates the maximum entropy rate $maxSR$ (`maxSr`) and initial entropy rate SR_0 (`SRo`) given a connected network.

Usage

```
getEntropyRate(gg)
```

Arguments

gg	igroph object

Details

The maximum entropy rate being calculated from the network's adjacency matrix:

$$maxSR = \sum_{i,j} p_{ij} = \frac{A_{ij}\nu_j}{\lambda\nu_i}$$

where ν and λ are the leading eigenvector and eigenvalue of the network adjacency matrix A respectively.

The initial configuration occurs when the entropy for each node is maximal. This can be calculated by setting the expression value for each gene/node in the network to be the same, and thus the maximal node entropy is dependent only on the node's degree k:

$$SR_0 = \frac{1}{N\bar{k}} \sum_j k_j \log k_i$$

where N here is the number of nodes and $\bar{k}$ the average node degree found in the network.

Value

list with values of maxSr and SRo

See Also

Other Entropy Functions: `calcEntropy()`, `getEntropy()`, `plotEntropy()`

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
ent <- getEntropyRate(karate)
```

---

getGNP *Generate random graph from reference*

Description

Function generates random G(n,p) Erdos-Renyi graph (`sample_gnp`) with the same number of vertices and edges as in in the reference graph gg.

Usage

```
getGNP(gg, ...)
```

Arguments

gg	reference graph
...	additional arguments to be passed to `sample_gnp`

Value

new instance of the random graph.

Examples

```
data(karate,package='igraphdata')
vcount(karate)
ecount(karate)
rg<- getGNP(karate)
vcount(rg)
ecount(rg)
```

getGraphCentralityECDF *Convert centrality matrix into ECDF*

Description

Convert centrality matrix into ECDF

Usage

```
getGraphCentralityECDF(m)
```

Arguments

m	centrality matrix from getCentralityMatrix invocation.

Value

list of several ecdf objects, corresponding to values in centrality matrix from getCentralityMatrix invocation.

See Also

getCentralityMatrix

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
m<-getCentralityMatrix(gg)
ecdfL<-getGraphCentralityECDF(m)
```

getIDs *Utility function to get vertex ids from vertex attributes The function obtain attribute values and check duplicates in it. It fails if any duplicate found.*

Description

Utility function to get vertex ids from vertex attributes The function obtain attribute values and check duplicates in it. It fails if any duplicate found.

Usage

```
getIDs(gg, idatt)
```

Arguments

gg	graph
idatt	attribute name

Value

idatt attribute values

---

getPA *Generate random graph from reference*

Description

The function generates random Barabasi-Albert graph (sample_pa) with the same vertex number as in the reference graph gg and the power specified by parameter pwr. If pwr is missing, we are trying to estimate pwr from the reference graph gg.

Usage

```
getPA(gg, pwr, ...)
```

Arguments

gg	reference graph
pwr	the power parameter for the sample_pa
...	additional parameters to be passed to the sample_pa

Value

new instance of the random graph.

Examples

```
data(karate,package='igraphdata')
vcount(karate)
ecount(karate)
rg<- getPA(karate,pwr=1.25)
vcount(rg)
ecount(rg)
```

`getRandomGraphCentrality` *Centrality measures for random graphs induced by input one*

Description

Generate a random graph that mimics the properties of the input graph and calls `getCentralityMatrix` to calculate all available vertex centrality measures. There are four different types of random graph to generate

Usage

```
getRandomGraphCentrality(
  gg,
  type = c("gnp", "pa", "cgnp", "rw"),
  power = NULL,
  weights = NULL,
  ...
)
```

Arguments

`gg`	template graph to mimic
`type`	type of random graph to generate: • gnp – G(n,p) Erdos-Renyi model (`sample_gnp`) • pa – Barabasi-Albert model (`sample_pa`) • cgnp – new random graph from a given graph by randomly a dding/removing edges (`sample_correlated_gnp`) • rw – new random graph from a given graph by rewiring 25% of edges preserving the degree distribution `sample_gnp`, `sample_correlated_gnp`, and `sample_pa`
`power`	optional argument of the power of the preferential attachment to be passed to `sample_pa`. If `power` is `NULL` the power of the preferential attachment will be estimated from `fitDegree` function.
`weights`	Possibly a numeric vector giving edge weights. If this is NULL and the graph has a weight edge attribute, then the attribute is used. If this is NA then no weights are used (even if the graph has a weight attribute).
`...`	other parameters passed to random graph generation functions

Value

matrix of random graph vertices centrality measure.

See Also

`getCentralityMatrix()` for explanation of the use of `weights`.

Examples

```
data(karate,package='igraphdata')
m<-getRandomGraphCentrality(karate,'pa',threads=1)
# to avoid repetitive costy computation of PowerLaw fit
# power parameter could be send explicitly:
pFit <- fitDegree( as.vector(igraph::degree(graph=karate)),
Nsim=10, plot=FALSE,threads=1)
pwr <- slot(pFit,'alpha')
m<-getRandomGraphCentrality(karate,'pa',power=pwr)
lpa<-lapply(1:5,getRandomGraphCentrality,gg=karate,type='pa',
            power=pwr,weights = NULL)
```

`getRobustness` *Calculate cluster robustness from consensus matrix*

Description

This function takes as argument a network (`gg`), the name of a clustering algorithm (`alg`) which can be found in the network, and a consensus matrix (`conmat`) generated from the clustering network. The function uses the consensus matrix to generate a measure of cluster robustness C_{rob} (`Crob`) for each cluster (`C`) using the R function `clrob`. Briefly, this is done by summing elements of the consensus matrix that are found in the same cluster, and dividing this by the total number of entries in the matrix:

$$C_{rob} = \frac{2}{C_n(C_n - 1)} \sum_{\substack{i,j \in I_C \\ i \leq j}} conmat_{i,j}$$

where I_C – indices of vertices of the cluster C, C_n is the number of nodes found inside the cluster C.

Usage

```
getRobustness(gg, alg, conmat)
```

Arguments

`gg`	igroph object
`alg`	clustering algorithm
`conmat`	consensus matrix

Value

data.frame that for each cluster `C` shows

- its size C_n (`Cn`),
- robustness C_{rob} (`Crob`) and
- robustness scaled to range between 0 and 1 (`CrobScaled`).

See Also

Other Robustness Functions: `makeConsensusMatrix()`

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
alg<-'louvain'
gg<-calcClustering(karate, alg = alg)
conmat<-makeConsensusMatrix(gg, N=100, mask = 10, alg = alg, type = 2)
clrob<-getRobustness(gg, alg = alg, conmat)
clrob
```

---

gofs *Goodnes of fit KS test*

Description

This is internal function and do not suppose to be called by user.

Usage

```
gofs(x, rate, model, sigma2 = NULL, countDATA = TRUE)
```

Arguments

`x`	steps along the Fe
`rate`	parameters of the sigmoid
`model`	fitted model
`sigma2`	noise strength
`countDATA`	should points to be counted

Value

list of `ks.test` values for each value in `rate`

law-class *Result of PawerLaw fit*

Description

Result of PawerLaw fit

Slots

`fit` `displ-class` result of power law fit.

`p` numeric.

`alpha` numeric degree of power-law.

`SDxmin` numeric bootstrap sd of Xmin.

`SDalpha` numeric bootstrap sd of alpha.

layoutByCluster *Calculate layout based upon membership*

Description

Function to split graph into clusters and layout each cluster independently..

Usage

```
layoutByCluster(gg, mem, layout = layout_with_kk)
```

Arguments

`gg`	graph to layout
`mem`	membership data.frame from `calcMembership`
`layout`	algorithm to use for layout

Value

Layout in a form of 2D matrix.

See Also

igraph::layout_

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
lay<-layoutByCluster(karate,mem)
#plot(karate,layout=lay)
```

layoutByRecluster *Calculate two-level layout from recluster matrix*

Description

Takes results of recluster and apply `layoutByCluster` to each

Usage

```
layoutByRecluster(gg, remem, layout = layout_with_kk)
```

Arguments

`gg`	graph to layout
`remem`	recluster result obtained by `calcReclusterMatrix` invocation
`layout`	one of the layout algorithms from `layout_`

Value

Layout in a form of 2D matrx.

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
remem<-calcReclusterMatrix(karate,mem,alg,10)
lay<-layoutByRecluster(karate,remem)
#plot(karate,layout=lay)
```

makeConsensusMatrix *Function to make random resampling consensus matrix in memory*

Description

Function to make random resampling consensus matrix in memory

Usage

```
makeConsensusMatrix(
  gg,
  N = 500,
  mask = 20,
  alg,
  type,
  weights = NULL,
```

```
  reclust = FALSE,
  Cnmax = 10
)
```

Arguments

gg	graph to perturb
N	number of perturbation steps
mask	percentage of elements to perturbe
alg	clustering alg.
type	edges (1) or nodes (2) to mask
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.
reclust	logical to decide wether to invoke reclustering via recluster
Cnmax	maximum size of the cluster in mem that will not be processed if reclustering is invoked

Details

Function to assess the robustness of network clustering. A randomisation study is performed apply the same clustering algorithm to N perturbed networks, and which returns the consensus matrix where each vertex pair is assigned the probability of belong to the same cluster. The inputted network is perturbed by randomly removing a mask percentage of edges (type=1) or vertices (type=2) from the network before clustering.

Value

consensus matrix of Nvert X Nvert

See Also

Other Robustness Functions: getRobustness()

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
alg<-'louvain'
gg<-calcClustering(karate, alg = alg)
conmat<-makeConsensusMatrix(gg, N=100, mask = 10, alg = alg, type = 2)
dim(conmat)
```

makeMembership *Create membership* data.frame *from graph for arbitrary annotation*

Description

Create membership data.frame from graph vertex attribute or vector of cluster names, IDs or indices. This function is simular to calcMembership but do not linked to clustering algorithm.

Usage

```
makeMembership(gg, membership)
```

Arguments

gg	igraph object to assign membership
membership	either name of the vertex attribute or vector of membership

Details

Any annotation coercible to factor could be converted to the membership data.frame. This function is useful, for example, to make layout with layoutByCluster.

Value

data.frame with two columns names and membership

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
m<-makeMembership(karate,rep(c(1,2),length.out=vcount(karate)))
head(m)
```

`markBowTie` *Calculates bow-tie decomposition and marks vertices with one of the following in the* `BowTie` *attribute:*

- *SCC – maximal strong connected component;*
- *IN – vertices not in SCC, but SCC is reachable from them;*
- *OUT – vertices not in SCC, but reachable from SCC;*
- *TU – vertices not in all three above, but reachable from IN and OUT is reachable from them (TUBES);*
- *IDR – vertices not in SCC, but they are reachable from IN and OUT is NOT reachable from them (INTENDRILS);*
- *ODR – vertices not is SCC, but they are NOT reachable from IN and OUT is reachable from them (OUTTENDRILS);*
- *OTR – all other vertices.*

Description

Algorithm proposed in:

Usage

```
markBowTie(g)
```

Arguments

g	graph to analyse

Details

"Bow-tie Decomposition in Directed Graphs" - Yang et al. IEEE (2011)

Value

graph with `BowTie` vertex attribute

`metlMatrix` *Convert sparce matrix into triplet* `data.frame`.

Description

For very large graphs handling adjacency-like matrices is difficult due to its sparse nature. This function convert sparse matrix into triplet `data.frame` with row and column indices and names, and cell value.

Usage

```
metlMatrix(sparceM)
```

Arguments

sparceM	sparce matrix to convert into triplet data.frame

Value

data.frame with three colums:

- i – row index;
- j – column index;
- x – cell value;
- Rname – i-th row name;
- Cname – j-th column name.

Examples

```
data(karate, package='igraphdata')
upgrade_graph(karate)
Ws <- as_adjacency_matrix(karate,type='both',attr='weight',sparse = TRUE)
mdf<-metlMatrix(Ws)
head(mdf)
```

---

normModularity *Calculates the normalised network modularity value.*

Description

Function to compare network Modularity of input network with networks of different size and connectivity.

Usage

```
normModularity(
  gg,
  alg = c("lec", "wt", "fc", "infomap", "louvain", "sgG1", "sgG2", "sgG5"),
  Nint = 1000,
  weights = NULL
)
```

Arguments

gg	graph object to analyze
alg	clustering algorithm
Nint	number of iterations
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.

Details

Used the normalised network modularity value Qm based on the previous studies by Parter et al., 2007, Takemoto, 2012, Takemoto, 2013, Takemoto and Borjigin, 2011, which was defined as:

$$Q_m = \frac{Q_{real} - Q_{rand}}{Q_{max} - Q_{rand}}$$

Where Q_{real} is the network modularity of a real-world signalling network and, Q_{rand} is the average network modularity value obtained from 10,000 randomised networks constructed from its real-world network. Q_{max} was estimated as: 1 - 1/M, where M is the number of modules in the real network.

Randomised networks were generated from a real-world network using the edge-rewiring algorithm (Maslov and Sneppen, 2002).

Value

normalized modularity value

References

Takemoto, K. & Kihara, K. Modular organization of cancer signaling networks is associated with patient survivability. Biosystems 113, 149–154 (2013).

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)

nm<-normModularity(gg, alg='louvain',Nint=10)
```

permute *Randomly shuffle annotations*

Description

This function is a convinience wrapper to `sample` with `replace= FALSE`

Usage

```
permute(GNS, N)
```

Arguments

GNS	annotation list to take data from
N	size of the sample

Value

random list of GNS values

Examples

```
permute(LETTERS, 15)
```

plotBridgeness *Plot Bridgeness values*

Description

Semi-local centrality measure (Chen et al., 2011) lies between 0 and 1 indicating whether protein is important globally or locally. By plotting Bridgeness against semi-local centrality we can categorises the influence each protein found in our network has on the overall network structure:

- Region 1, proteins having a 'global' rather than 'local' influence in the network (also been called bottle-neck bridges, connector or kinless hubs (0<Sl<0.5; 0.5<Br<1).
- Region 2, proteins having 'global' and 'local' influence (0.5<Sl<1, 0.5<Br<1).
- Region 3, proteins centred within the community they belong to, but also communicating with a few other specific communities (0<Sl<0.5; 0.1<Br<0.5).
- Region 4, proteins with 'local' impact , primarily within one or two communities (local or party hubs, 0.5<Sl<1, 0<Br<0.5).

Usage

```
plotBridgeness(
  gg,
  alg,
  VIPs,
  Xatt = "SL",
  Xlab = "Semilocal Centrality (SL)",
  Ylab = "Bridgeness (B)",
  bsize = 3,
  spsize = 7,
  MainDivSize = 0.8,
  xmin = 0,
  xmax = 1,
  ymin = 0,
  ymax = 1,
  baseColor = "royalblue2",
  SPColor = "royalblue2"
)
```

Arguments

gg	igraph object with bridgenes values stored as attributes, after call to calcBridgeness
alg	clustering algorithm that was used to calculate bridgeness values
VIPs	list of 'specical' genes to be marked on the plot
Xatt	name of the attribute that stores values to be used as X-axis values. By default SL for semi-local centrality
Xlab	label for the X-axis
Ylab	label for the Y-axis
bsize	point size for genes
spsize	point size for 'specical' genes
MainDivSize	size of the line for the region separation lines
xmin	low limit for X-axis
xmax	upper limit for X-axis
ymin	low limit for Y-axis
ymax	upper limit for Y-axis
baseColor	basic color for genes
SPColor	colour highlighting any 'specical' genes

Value

ggplot object with plot

Examples

```
karate <- make_graph("Zachary")
# We need vertex ID in the 'name' attribute of the vertex
V(karate)$name<-c(LETTERS,letters)[1:vcount(karate)]
set.seed(100)
gg <- calcClustering(karate, 'louvain')
gg <- calcCentrality(gg)
cnmat <- makeConsensusMatrix(gg, N=10, alg = 'louvain', type = 2, mask = 10)
gg<-calcBridgeness(gg, alg = 'louvain', cnmat)
plotBridgeness(gg,alg = 'louvain',VIPs=c("Mr Hi","John A"))
```

`plotEntropy` *Plot graph entropy values versus vertex degree for each perturbed vertex value.*

Description

Following procedure described in (Teschendorff et al., 2015), all vertexes are artificially assigned a uniform weight then sequentially perturbed with the global entropy rate (`SRprime`) after each protein's perturbation being calculated by `getEntropy` function.

Usage

```
plotEntropy(SRprime, subTIT = "Entropy", SRo = NULL, maxSr = NULL)
```

Arguments

`SRprime`	results of `getEntropy` invocation
`subTIT`	entropy axis label
`SRo`	initial entropy rate SR_0, results of `getEntropyRate` invocation
`maxSr`	the maximum entropy rate $maxSR$, results of `getEntropyRate` invocation

Details

This function plot `SRprime` against the log of the protein's degree. In case of scale-free or approximate scale-free topologies, we see a clear bi-modal response between over-weighted vertices and their degree and an opposing bi-phasic response in under-weighted vertices and their degrees.

If `maxSr` or `SRo` is set to their default value `NULL` `getEntropyRate` will be called and returned values will be used in the following calculations. As `maxSr` is required for `SRprime` calculation by `getEntropy` using explicit values could save some time in the case of large network.

Value

ggplot2 object with diagram

See Also

getEntropy()

Other Entropy Functions: calcEntropy(), getEntropy(), getEntropyRate()

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.csv", package = "BioNAR")
tbl <- read.csv(file, sep="\t")
gg <- buildNetwork(tbl)
gg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(gg)$name == '80273')
paste(V(gg)$GeneName[idx], 'GRPEL1')
ent <- getEntropyRate(gg)
SRprime <- getEntropy(gg, maxSr = NULL)
plotEntropy(SRprime, subTIT = "Entropy", SRo = ent$SRo, maxSr = ent$maxSr)
```

---

plotRatio *Plot fraction of enriched communities*

Description

Plot fraction of enriched communities

Usage

```
plotRatio(
  x,
  desc = "",
  anno = "",
  LEGtextSize = 1.5,
  LEGlineSize = 4,
  type = NULL
)
```

Arguments

x	enrichment statistics
desc	plot subtitle
anno	name of annotation used
LEGtextSize	size of the text
LEGlineSize	width of the line
type	type of the plot

Value

ggplot object

plotSigmoid *Plot results of the sigmoid fit*

Description

Plot results of the sigmoid fit

Usage

```
plotSigmoid(x, rates, model, alg = "", pv = 0)
```

Arguments

x	steps along the Fe
rates	parameters of the sigmoid
model	fitted model
alg	name of the clustering algorithm
pv	Kolmogorov-Smirnov test's p-value

Value

ggplot object with sigmoid fit plot

PPI_Presynaptic.csv *Table of protein protein interactions for presynaptic compartment*

Description

Protein-protein interactions (PPIS) for presynaptic compartment, extracted from Synaptome.db, in a csv form. Columns A and B correspond to Entrez IDs for interacting proteins A and B (node names); column We contains the edge weights, if available.

See Also

buildNetwork

PPI_Presynaptic.gml *PPI graph for presynaptic compartment*

Description

Protein-protein interactions (PPIS) for presynaptic compartment, extracted from Synaptome.db, and saved in a graph format. Graph contains node attributes, such as names (Entrez IDs), Gene Names, disease association (TopOntoOVG, TopOntoOVGHDOID), annotation with schizophrenia-related genes (Schanno (v/c), function annotation from GO (GOBPID, GOBP, GOMFID, GOMF, GOCCID, GOCC), centrality measures (DEG - degree, BET - betweenness, CC - clustering coefficient, SL - semilocal centrality, mnSP - mean shortest path, PR - page rank, sdSP - standard deviation of the shortest path), and clustering memberships for 8 clustering algorithms (lec, wt, fc, infomap, louvain, sgG1, sgG2, sgG5)

prepareGDA *Function to return vertex annotation from a graph in the Vertex Annotation form and format it for further analysis.*

Description

Function to return vertex annotation from a graph in the Vertex Annotation form and format it for further analysis.

Usage

```
prepareGDA(gg, name)
```

Arguments

gg	igraph object to take annotation from
name	name of the vertex attribute that contains annotation. If graph has no such vertex attribute an error is thrown..

Value

escaped annotation in Vertex Annotation form

See Also

getAnnotationVertexList

escapeAnnotation

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
paste(V(agg)$GeneName[idx], 'GRPEL1')
gda<-prepareGDA(agg, 'TopOntoOVGHDOID')
gda<-prepareGDA(agg, 'TopOntoOVGHDOID')
head(gda)
```

PresynAn.csv *Presynaptic genes specific functional annotation*

Description

Presynaptic genes functional annotation derived from Boyken at al. (2013) doi:10.1016/j.neuron.2013.02.027. The table has columns: the first containing functional group ID terms, the second - gene functional group description terms, third - gene Human Entrez Ids; in csv format

See Also

annotatePresynaptic

---

recluster *Hierarchical graph clustering*

Description

Function reads in a graph GG with cluster membership stored in vertex attribute ALGN, and reapplies the clustering algorithm ALGN to all clusters larger than CnMAX

Usage

```
recluster(GG, ALGN, CnMAX, weights = NULL)
```

Arguments

GG	graph to cluster
ALGN	algorithm to apply
CnMAX	maximum size of the cluster in mem that will not be processed
weights	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the spectral clustering.

Value

remembership matrix, that contains vertex ID membership and result of reclustering

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
remem<-calcReclusterMatrix(karate,mem,alg,10)
```

`removeVertexTerm` *Remove vertex property.*

Description

Remove vertex property.

Usage

```
removeVertexTerm(GG, NAME)
```

Arguments

GG	igraph object
NAME	name of the vertex property to remove

Value

igraph object with attribute removed

Examples

```
data(karate, package='igraphdata')
upgrade_graph(karate)
vertex_attr_names(karate)
m<-removeVertexTerm(karate, 'color')
vertex_attr_names(m)
```

runPermDisease *Calculate disease-disease pair overlaps on permuted network to estimate its statistical significance*

Description

Function to calculate the disease-pair overlap characteristics of an inputted network, before applying Nperm permutations on the disease annotations of #' type "random" or "binned" permute. From the permuted networks the function estimates the significance of disease overlap: p-value, Bonferoni-adjusted p-value, and q-value in the Disease_overlap_sig. The function also compares the average disease separation between inputted and permuted networks, and calculates its significance using the Wilcox test and store. Significance of disease-pair overlap and disease separation results are stored in the matrix Disease_location_sig.

Usage

```
runPermDisease(
  gg,
  name,
  diseases = NULL,
  Nperm = 100,
  permute = c("random", "binned"),
  alpha = c(0.05, 0.01, 0.001)
)
```

Arguments

gg	interactome network as igraph object
name	name of the attribute that stores disease annotation
diseases	list of diseases to match
Nperm	number of permutations to apply
permute	type of permutations. random – annotation is randomly shuffled, binned – annotation is shuffled in a way to preserve node degree-annotation relationship by degreeBinnedGDAs.
alpha	statistical significance levels

Details

Run with care, as large number of permutations could require a lot of memory and be timeconsuming.

Value

list of two matrices: Disease_overlap_sig gives s tatistics for each pair of disease, and Disease_location_sig gives intra-disease statistics

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
paste(V(agg)$GeneName[idx], 'GRPEL1')
r <- runPermDisease(
agg,
name = "TopOntoOVGHDOID",
diseases = c("DOID:10652", "DOID:3312", "DOID:12849", "DOID:1826"),
Nperm = 10,
alpha = c(0.05, 0.01, 0.001))
r$Disease_location_sig
```

---

sampleDegBinnedGDA *Function to randomly shuffle vertex annotation terms, whilst preserving the vertex degree originally found with that annotation term..*

Description

Function to randomly shuffle vertex annotation terms, whilst preserving the vertex degree originally found with that annotation term..

Usage

```
sampleDegBinnedGDA(org.map, term)
```

Arguments

org.map	degree-annotation mapping returned by degreeBinnedGDAs
term	annotation term to shuffle

Value

vertex IDs to assign term in shuffled annotation

See Also

degreeBinnedGDAs

Examples

```
file <- system.file("extdata", "PPI_Presynaptic.gml", package = "BioNAR")
gg <- igraph::read_graph(file, format="gml")
agg<-annotateGeneNames(gg)
# due to error in org.Hs.eg.db we have to manually check annotation of one node
idx <- which(V(agg)$name == '80273')
```

```
paste(V(agg)$GeneName[idx], 'GRPEL1')
gda<-prepareGDA(agg, 'TopOntoOVGHDOID')
diseases<-getAnnotationList(gda)
m<-degreeBinnedGDAs(agg, gda, diseases)
sampleDegBinnedGDA(m, diseases[1])
```

`sampleGraphClust` *Perturbe graph and calculate its clustering*

Description

Function will mask `mask` a percentage of edges (`type=1`) or vertices (`type=2`) from the network, find the largest connected component of the masked network and cluster it. The clustering results are stored in a three column matrix: the first column contains the vertex IDs of input network; the second column the vertex IDs of the subsampled network, or -1 if the vertex has been masked; the third column the cluster membership of subsampled network, or -1 if vertex has been masked.

Usage

```
sampleGraphClust(
  gg,
  mask = 20,
  alg,
  type,
  weights = NULL,
  reclust = FALSE,
  Cnmax = 10
)
```

Arguments

`gg`	graph
`mask`	percentage of elements to perturbe
`alg`	clustering alg.
`type`	edges=>1 or nodes=>2 to mask
`weights`	The weights of the edges. It must be a positive numeric vector, NULL or NA. If it is NULL and the input graph has a 'weight' edge attribute, then that attribute will be used. If NULL and no such attribute is present, then the edges will have equal weights. Set this to NA if the graph was a 'weight' edge attribute, but you don't want to use it for community detection. A larger edge weight means a stronger connection for this function. The weights value is ignored for the `spectral` clustering.
`reclust`	logical to decide whether to invoke reclustering via `recluster`
`Cnmax`	maximum size of the cluster in `mem` that will not be processed if reclustering is invoked

Details

This is internal function and not supposed to be calle by end user.

Value

list of Nx3 matrices

Examples

```
data(karate,package='igraphdata')
alg<-'louvain'
mem<-calcMembership(karate,alg = alg)
smpl<-BioNAR:::sampleGraphClust(karate,mask=10,alg,type=2)
```

SCH_flatfile.csv *Schizopherina related synaptic gene functional annotation.*

Description

Annotation, manually curated from an external file: Lips et al., (2012) doi:10.1038/mp.2011.117.The table has columns: the first containing gene Human Entrez IDs, the second gene functional group ID terms, the third gene functional group description terms; in csv format

See Also

annotateSCHanno

---

summaryStats *Calculate summary statistics from enrichment table*

Description

Calculate summary statistics from enrichment table

Usage

```
summaryStats(RES, ALPHA, usePadj = FALSE, FeMAX = 0, FcMAX = 0)
```

Arguments

RES	enrichment results data.frame
ALPHA	p-value cut-off
usePadj	logical, wether to use plain or adjusted p-value
FeMAX	max of the FE
FcMAX	max of the FC

Value

list of `data.frame`

`unescapeAnnotation` *Unescape annotation strings*

Description

Function to remove all escape characters from annotation strings (opposite to escapeAnnotation).

Usage

```
unescapeAnnotation(annVec, col = COLLAPSE, esc = ESC)
```

Arguments

`annVec`	vector of annotation strings
`col`	list separator character within annotation string
`esc`	escape character

Details

NOTE: spaces are treated as regular characters, no trimming is applied before or after escaping.

Value

vector of annotation strings with removed escape characters

See Also

escapeAnnotation

Examples

```
annVec<-apply(matrix(letters, ncol=13), 2, paste, collapse=';')
escVec<-escapeAnnotation(annVec, ';', '|')
cbind(annVec, escVec, unescapeAnnotation(escVec, ';', '|'))
```

zeroNA *Auxiliary function to replace NAs with zeros.*

Description

Auxiliary function to replace NAs with zeros.

Usage

```
zeroNA(x)
```

Arguments

x	matrix or vector to process

Value

matrix or vector with NAs replaced by zero.

Examples

```
x<-matrix(NA,nrow = 3,ncol = 3)
zeroNA(x)
```

Index

∗ **Entropy Functions**
  calcEntropy, 23
  getEntropy, 46
  getEntropyRate, 47
  plotEntropy, 63
∗ **Robustness Functions**
  getRobustness, 52
  makeConsensusMatrix, 55
∗ **diseasome**
  diseasome, 31
∗ **file**
  flatfile.go.BP.csv, 36
  flatfile.go.CC.csv, 36
  flatfile.go.MF.csv, 37
  flatfile_human_gene2HDO.csv, 37
  PPI_Presynaptic.csv, 65
  PPI_Presynaptic.gml, 66
  PresynAn.csv, 67
  SCH_flatfile.csv, 72
∗ **graphs**
  diseasome, 31
∗ **internal**
  buildConsensusMatrix, 15

addEdgeAtts, 4
annotateGeneNames, 4
annotateGoBP, 5, *36*
annotateGoCC, 6, *36*
annotateGoMF, 7, *37*
annotateGOont, 8
annotateInterpro, 9
annotatePresynaptic, 9, *67*
annotateSCHanno, 10, *72*
annotateTopOntoOVG, 11, *37*
annotateVertex, 12, *13*
applpMatrixToGraph, 13, *18*
as_undirected, *41*

betweenness, *40*
BioNAR, 14
BioNAR-package *(*BioNAR*)*, 14
buildConsensusMatrix, 15
buildNetwork, 15, *65*

calcAllClustering, 16
calcBridgeness, 17, *62*
calcCentrality, 18
calcCentralityExternalDistances, 19
calcCentralityInternalDistances, 20
calcClustering, *16*, 21
calcDiseasePairs, 22
calcEntropy, 23, *47, 48*, *64*
calcMembership, 24, *54*, *57*
calcReclusterMatrix, 25, *55*
calcSparsness, 26
clrob, *52*
cluster_fast_greedy, *41*
cluster_infomap, *41*
cluster_leading_eigen, *41*
cluster_louvain, *41*
cluster_spinglass, *41*
cluster_walktrap, *41*
clusteringSummary, 27
clusterORA, 28
communities, *42*
compMembership, 29

degreeBinnedGDAs, *22*, 30, *69, 70*
diseasome, 31
dist, *20*

escapeAnnotation, 32
evalCentralitySignificance, 33

findLCC, 34
fitDegree, 34, *51*
fitSigmoid, 35
flatfile.go.BP.csv, 36
flatfile.go.CC.csv, 36
flatfile.go.MF.csv, 37

flatfile_human_gene2HDO.csv, 37

getAnnotationList, 37
getAnnotationVertexList, 38
getBridgeness, *17*, 39
getCentralityMatrix, *18, 19*, 40, *49*, *51*
getCentralityMatrix(), *18*, *51*
getClustering, *21*, *24*, *29*, *40*, 41
getClusterSubgraphByID, 42
getCommunityGraph, 43
getDiseases, 44
getDType, 44
getDYNAMO, 45
getEntropy, *23, 24*, 46, *48*, *63, 64*
getEntropy(), *24*, *64*
getEntropyRate, *24*, *47*, 47, *63, 64*
getGNP, 48
getGraphCentralityECDF, 49
getIDs, 49
getPA, 50
getRandomGraphCentrality, *19, 20*, 51
getRobustness, 52, *56*
ggplot, *62*, *65*
gofs, 53
graph, *31*
graph_from_data_frame, *15*

keys, *5*, *8*
keytypes, *5*, *8*
ks.test, *53*

law-class, 54
layout_, *55*
layoutByCluster, 54, *57*
layoutByRecluster, 55
legend(), *35*

makeConsensusMatrix, *53*, 55
makeMembership, 57
markBowTie, 58
match.arg, *21*, *42*
metlMatrix, 58
modularity, *27*

normModularity, 59

org.Hs.eg.db, *4*, *8*

page_rank, *40*
permute, 61
plotBridgeness, 61
plotEntropy, *24*, *47, 48*, 63
plotRatio, 64
plotSigmoid, 65
PPI_Presynaptic.csv, 65
PPI_Presynaptic.gml, 66
prepareGDA, *30*, 66
PresynAn.csv, 67

recluster, *56*, 67, *71*
removeVertexTerm, 68
runPermDisease, 69

sample, *61*
sample_correlated_gnp, *51*
sample_gnp, *48*, *51*
sample_pa, *50, 51*
sampleDegBinnedGDA, 70
sampleGraphClust, *15*, 71
SCH_flatfile.csv, 72
spectral_igraph_communities, *41*
summaryStats, *35*, 72
summaryStats(), *36*

unescapeAnnotation, 73

vcount, *27*

zeroNA, 74