

# Package ‘BioNAR’

February 2, 2023

**Title** Biological Network Analysis in R

**Version** 1.0.0

**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** 7.2.1

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

**Imports** stringr, synaptome.db, clusterCons, fgsea, grid, methods, scales, AnnotationDbi, dplyr, GO.db, org.Hs.eg.db, rSpectral, WGCNA, ggplot2, ggrepel

**RdMacros** Rdpack

**Suggests** knitr, rmarkdown, igraphdata, testthat (>= 3.0.0), vdiffR, devtools, pander, plotly, randomcoloR

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**biocViews** Software, GraphAndNetwork, Network

**git\_url** <https://git.bioconductor.org/packages/BioNAR>

**git\_branch** RELEASE\_3\_16

**git\_last\_commit** f9c6c3c

**git\_last\_commit\_date** 2022-11-01

**Date/Publication** 2023-02-02

**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>

## R topics documented:

addEdgeAtts . . . . .	3
annotateGeneNames . . . . .	4
annotateGoBP . . . . .	5
annotateGoCC . . . . .	6
annotateGoMF . . . . .	6
annotateGOont . . . . .	7
annotateInterpro . . . . .	8
annotatePresynaptic . . . . .	9
annotateSCHanno . . . . .	9
annotateTopOntoOVG . . . . .	10
annotateVertex . . . . .	11
applpMatrixToGraph . . . . .	12
BioNAR . . . . .	13
buildConsensusMatrix . . . . .	13
buildFromSynaptomeByEntrez . . . . .	14
buildFromSynaptomeGeneTable . . . . .	14
buildNetwork . . . . .	15
calcAllClustering . . . . .	16
calcBridgeness . . . . .	16
calcCentrality . . . . .	17
calcCentralityExternalDistances . . . . .	18
calcCentralityInternalDistances . . . . .	19
calcClustering . . . . .	20
calcDiseasePairs . . . . .	21
calcEntropy . . . . .	22
calcMembership . . . . .	23
calcReclusterMatrix . . . . .	24
calcSparsness . . . . .	24
clusteringSummary . . . . .	25
clusterORA . . . . .	26
degreeBinnedGDAs . . . . .	27
diseasome.rda . . . . .	28
escapeAnnotation . . . . .	28
evalCentralitySignificance . . . . .	29
findLCC . . . . .	30
fitDegree . . . . .	31
flatfile.go.BP.csv . . . . .	32
flatfile.go.CC.csv . . . . .	32
flatfile.go.MF.csv . . . . .	32
flatfile_human_gene2HDO.csv . . . . .	33
getAnnotationList . . . . .	33

getAnnotationVertexList . . . . .	34
getBridgeness . . . . .	35
getCentralityMatrix . . . . .	36
getClustering . . . . .	37
getClusterSubgraphByID . . . . .	38
getCommunityGraph . . . . .	38
getDiseases . . . . .	39
getDType . . . . .	39
getEntropy . . . . .	40
getEntropyRate . . . . .	41
getGraphCentralityECDF . . . . .	41
getRandomGraphCentrality . . . . .	42
getRobustness . . . . .	43
law-class . . . . .	44
layoutByCluster . . . . .	44
layoutByRecluster . . . . .	45
makeConsensusMatrix . . . . .	45
normModularity . . . . .	46
permute . . . . .	48
plotBridgeness . . . . .	48
plotEntropy . . . . .	50
PPI_Presynaptic.csv . . . . .	51
PPI_Presynaptic.gml . . . . .	51
prepareGDA . . . . .	51
PresynAn.csv . . . . .	52
recluster . . . . .	52
removeVertexTerm . . . . .	53
runPermDisease . . . . .	54
sampleDegBinnedGDA . . . . .	55
sampleGraphClust . . . . .	56
SCH_flatfile.csv . . . . .	57
unescapeAnnotation . . . . .	57
zeroNA . . . . .	58

**Index****59**


---

 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](http://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](http://org.Hs.eg.db)  
 keytype             type of IDs stored in the name vertex attribute, by default ENTREZID is assumed.

**Details**

If vertex name attribute 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 attribute could be provided in the keytype attribute.

If for some vertices name attribute 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)
```

---

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)
```

## Arguments

gg	graph to update
annoF	annotation matrix in Pair form

## 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	<i>Add GO CC annotation to the graph vertices</i>
--------------	---

---

### 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)
```

### Arguments

gg	graph to update
annoF	annotation matrix in Pair form

### 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	<i>Add GO MF annotation to the graph vertices</i>
--------------	---

---

### 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)
```

**Arguments**

```
gg          graph to update
annoF      annotation matrix in Pair form
```

**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 GeneOntology 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")
```

**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 attribute 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 attribute could be provided in the `keytype` attribute.

If for some vertices name attribute 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	<i>Add InterPro Family and Domain annotation to the graph vertices</i>
------------------	--

---

**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)
```

**Arguments**

<code>gg</code>	graph to update
<code>annoF</code>	family annotation matrix in Pair form
<code>annoD</code>	domain annotation matrix in Pair form

**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 et al. (2013) doi:10.1016/j.neuron.2013.02.027 and add them to attributes PRESYNAPTIC.

### Usage

```
annotatePresynaptic(gg, anno)
```

### Arguments

gg	graph to update
anno	annotation matrix in Pair form

### 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)
```

**Arguments**

gg	igraph object to annotate
anno	annotation matrix in Pairs form

**Details**

References:

1. Lips E, Cornelisse L, Toonen R, Min J, Hultman C, the International Schizophrenia 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**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
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)
```

**Arguments**

```
gg          igraph object to annotate
dis         annotation matrix in Pairs form
```

**Value**

annotated igraph object

**See Also**

```
getAnnotationVertexList
```

**Examples**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
# 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	<i>Generic annotation function</i>
----------------	------------------------------------

---

**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 first containing vertex name IDs, and the second the vertex annotation value.

**Usage**

```
annotateVertex(gg, name, values)
```

**Arguments**

```
gg          igraph object to annotate
name        name of the attribute
values      annotation data.frame
```

**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
```

---

`aplpMatrixToGraph`     *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**

```
aplpMatrixToGraph(gg, m)
```

**Arguments**

<code>gg</code>	igraph object
<code>m</code>	matrix of values to be applied as vertex attributes. matrix should contain column "ID" to map value to the vertex.

**Details**

Unlike [annotateVertex](#), which is able to collapse multiple annotation terms, this function assumes that vertex ID values are unique in the `m` matrix.

**Value**

modified igraph object

**See Also**

annotateVertex

**Examples**

```
g1 <- make_star(10, mode="undirected")
V(g1)$name <- letters[1:10]
m<-cbind(ID=letters[1:10],capital=LETTERS[1:10])
g1<-BioNAR::app1pMatrixToGraph(g1,m)
V(g1)$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]

---

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 clustering matrices obtained from the [sampleGraphClust](#)

**Value**

consensus matrix

`buildFromSynaptomeByEntrez`

*Utility function to create network from [synaptome.db](#) data*

**Description**

Utility function to create network from [synaptome.db](#) data

**Usage**

```
buildFromSynaptomeByEntrez(entrez)
```

**Arguments**

entrez                vector of EntrezIDs for network vertices

**Value**

largest connected component of network defined by the gene list

**Examples**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments()$Name)
geneTable<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(geneTable$HumanEntrez)
```

`buildFromSynaptomeGeneTable`

*Utility function to create network from [synaptome.db](#) data*

**Description**

Utility function to create network from [synaptome.db](#) data

**Usage**

```
buildFromSynaptomeGeneTable(geneTable)
```

**Arguments**

geneTable      data.frame described in [getGenesByID](#)

**Value**

largest connected component of network defined by the gene table

**Examples**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments())$Name)
geneTable<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeGeneTable(geneTable)
```

---

buildNetwork	<i>Build network from data.table</i>
--------------	--------------------------------------

---

**Description**

Wrapper for [graph.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 if it exists.

**Usage**

```
buildNetwork(ff, kw = NA)
```

**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

**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	<i>Calculate memberships for all clustering algorithms and store them on the graph vertices.</i>
-------------------	--

---

### 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)
```

### Arguments

gg                    graph for analysis

### 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	<i>Helper function that uses <a href="#">getBridgeness</a> to calculate graph node bridgeness values for selected algorithm and consensus matrix and save them as a graph attribute BRIDGENESS.&lt;alg&gt; with &lt;alg&gt; replaced by the selected algorithm name.</i>
----------------	--

---

### 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)
data(karate, package='igraphdata')
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	<i>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.</i>
----------------	---

---

**Description**

A wrapper function that first calls [getCentralityMatrix](#), to calculate all vertex centrality measures, and then [applMatrixToGraph](#) to store each centrality result as a new vertex attribute in the graph.

**Usage**

```
calcCentrality(gg)
```

**Arguments**

gg	igraph object
----	---------------

**Value**

modified igraph object

**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 <a href="#">getCentralityMatrix</a>
l	list of permuted matrix, for example centrality obtained by invocation <a href="#">getRandomGraphCentrality</a>
keepOrder	if FALSE values 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 <a href="#">getRandomGraphCentrality</a>
keepOrder	if FALSE values will be sorted before distance calculations
dist	methods available from <a href="#">dist</a> 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	<i>Calculate community membership for given clustering algorithm and store the results as new vertex attributes in the graph..</i>
----------------	--

---

**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)
```

**Arguments**

gg	igraph object to cluster
alg	algorithm to apply

**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**

```
data(karate, package='igraphdata')
g<-calcClustering(karate, 'louvain')
vertex_attr_names(g)
graph_attr(g, 'louvain')
```

---

calcDiseasePairs	<i>Calculate each disease-disease pair overlap given a list of disease terms.</i>
------------------	---

---

**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 <a href="#">degreeBinnedGDAs</a> .

**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)
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**

Calculate the graph entropy for each perturbed vertex, and save the results as new vertex attributes in the graph. Given a PPI network, the function will calculate the graph entropy for each vertex over- and under-expressed, where the user can set what over- and under-expressed values each vertex takes (Teschendorff et al, 2014).

**Usage**

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

**Arguments**

gg	igraph object
maxSr	maxSr value, 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.

**Value**

graph with SR\_UP and SR\_DOWN vertex attributes storing the graph entropy values with over- or under-expressing each vertex.

## Examples

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
gg<-annotateGeneNames(gg)
gg<- calcEntropy(gg)
```

---

calcMembership	<i>Calculate cluster memberships for the graph.</i>
----------------	---

---

## Description

Calculates the clustering membership for each of the 10 clustering algorithms defined in function [getClustering](#)

## Usage

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

## Arguments

gg	igraph object to cluster
alg	algorithm name

## Value

data.frame with columns names and membership

## See Also

[getClustering](#)

## Examples

```
data(karate, package='igraphdata')
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, keepSplit = FALSE)
```

### Arguments

<code>gg</code>	graph to cluster
<code>mem</code>	data.frame with previous level clustering results
<code>alg</code>	algorithm to apply
<code>CnMAX</code>	maximum size of the cluster in <code>mem</code> that will not be processed
<code>keepSplit</code>	logical, whether to keep previous membership in the output matrix

### Value

membership 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 ( $N$ ):  $E/\text{binom}(N,2)$

### Usage

```
calcSparsness(gg)
```



**Arguments**

`gg` graph to evaluate

**Value**

sparsness value

**Examples**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
calcSparsness(gg)
```

---

clusteringSummary      *Matrix of cluster characteristics*

---

**Description**

Function to calculate basic summary statistics after apply clustering algorithm:

- 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 singletons (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: the cluster ID, the cluster size, overlap of annotation terms in cluster, p.value of enrichment using the Hypergeometric test, adjusted p.value Bonferroni correction (BH).

**Usage**

```
clusterORA(g, alg, name, vid = "name", alpha = 0.1, 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 ;

**Value**

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

- pathway – name of the enriched term as in 'names(pathway)';
- pval – an enrichment p-value from hypergeometric test;
- padj – a BH-adjusted p-value;

- overlap – size of the overlap;
- size – size of the gene set;
- leadingEdge – vector with overlapping genes.
- cl – cluster ID

### 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=get.vertex.attribute(g, 'TopOntoOVGHDOID'),
Term=get.vertex.attribute(g, 'TopOntoOVG')))
rr<-merge(andf, res, by.y='pathway', by.x='ID')
rr[order(rr$cl), ]
```

---

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 <a href="#">prepareGDA</a>
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)
gda<-prepareGDA(agg, 'TopOntoOVGHDOID')
m<-degreeBinnedGDAs(agg, gda, getAnnotationList(gda))
c(dim(m), vcount(agg), length(getAnnotationList(gda)))
head(m)
```

---

diseasome.rda

*Barabasi's Diseasome Network*


---

**Description**

In the paper Goh et 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.

**Details**

*Diseasome* 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	<i>Fit Power Law to degree distribution.</i>
-----------	--

---

### 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	height of the plot in ptx
legpos	position of the legend @seealsolegend
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)
```

---

flatfile.go.BP.csv      *Annotation from Gene Ontology Biological Process (GO\_BP)*

---

### Description

Annotation, downloaded from Gene Ontology for Biological Process domain. The table has columns: the first containing 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 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 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

[annotateTopOnto0VG](#)

---

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	<i>Calculate bridginess from consensus matrix</i>
---------------	---

---

## 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](https://doi.org/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)
data(karate, package='igraphdata')
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)
```

### Arguments

gg                    igraph object

### Value

matrix with following columns:

- ID - vertex ID
- DEG - degree
- BET - betweenness
- CC - clustering coefficient
- SL - semilocal centrality
- mnSP - mean shortest path
- PR - page rank
- sdSP - standard deviation of the shortest path

### Examples

```
library(synaptome.db)
cid<-match('Presynaptic',getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
m<-getCentralityMatrix(gg)
```

---

getClustering	<i>Get clustering results for the graph.</i>
---------------	--

---

## Description

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

- lec – leading eigenvector community (version of [leading.eigenvector.community](#));
- wt – walktrap community [walktrap.community](#);
- fc – fastgreedy community [fastgreedy.community](#);
- infomap – infomap community [cluster\\_infomap](#);
- louvain – cluster\_louvain [cluster\\_louvain](#);
- sgG1 – spin-glass model and simulated annealing clustering (version of [spinglass.community](#) with spins=500 and gamma=1);
- sgG2 – spin-glass model and simulated annealing clustering (version of [spinglass.community](#) with spins=500 and gamma=2);
- sgG5 – spin-glass model and simulated annealing clustering (version of [spinglass.community](#) 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")  
)
```

## Arguments

gg	igraph object to cluster
alg	clustering algorithm name

## 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

<code>clID</code>	cluster ID to extract
<code>gg</code>	graph to analyze
<code>mem</code>	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

<code>gg</code>	graph to convert
<code>membership</code>	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()
```

---

getEntropy	<i>Calculates vertex perturbation graph entropy.</i>
------------	--

---

**Description**

Calculates vertex perturbation graph entropy.

**Usage**

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

**Arguments**

gg	igraph object
maxSr	maxSr value, 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.

**Value**

matrix containing Gene: Entrez ID, Name, Degree and Graph Entropy values when gene is expressed up and down.

**Examples**

```
library(synaptome.db)
cid<-match('Presynaptic', getCompartments()$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
gg<-annotateGeneNames(gg)
e<- getEntropy(gg)
```



---

getEntropyRate      *Calculate parameters for Entropy plot and calculations.*

---

**Description**

Calculates vertex perturbed graph entropy values and parameters.

**Usage**

```
getEntropyRate(gg)
```

**Arguments**

gg                    igraph object

**Value**

list with values of maxSr and SRo

**Examples**

```
data(karate, package='igraphdata')
ent <- getEntropyRate(karate)
```

---

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**

```
library(synaptome.db)
cid<-match('Presynaptic',getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
m<-getCentralityMatrix(gg)
ecdfL<-getGraphCentralityECDF(m)
```

---

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,
  ...
)
```

**Arguments**

gg	template graph to mimic
type	type of random graph to generate: <ul style="list-style-type: none"> <li>• gnp – G(n,p) Erdos-Renyi model (<a href="#">sample_gnp</a>)</li> <li>• pa – Barabasi-Albert model (<a href="#">sample_pa</a>)</li> <li>• cgnp – new random graph from a given graph by randomly adding/removing edges (<a href="#">sample_correlated_gnp</a>)</li> <li>• rw – new random graph from a given graph by rewiring 25% of edges preserving the degree distribution</li> </ul>
power	optional argument of the power of the preferential attachment to be passed to <a href="#">sample_pa</a> . If power is NULL the power of the preferential attachment will be estimated from <a href="#">fitDegree</a> function.
...	other parameters passed to random graph generation functions <a href="#">sample_gnp</a> , <a href="#">sample_correlated_gnp</a> , and <a href="#">sample_pa</a>

**Value**

matrix of random graph vertices centrality measure.

**Examples**

```
data(karate, package='igraphdata')
m<-getRandomGraphCentrality(karate, 'pa', threads=1)
# to avoid repetitive costly 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)
```

---

getRobustness

*Calculate cluster robustness from consensus matrix*

---

**Description**

Calculate cluster robustness from consensus matrix

**Usage**

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

**Arguments**

gg	igraph object
alg	clustering algorithm
conmat	consensus matrix

**Value**

data.frame that for each cluster C shows its size  $C_n$ , robustness  $C_{rob}$  and robustness scaled to range between 0 and 1.  $C_{robScaled}$ .

**Examples**

```
data(karate, package='igraphdata')
alg<-'louvain'
gg<-calcClustering(karate, alg = alg)
conmat<-makeConsensusMatrix(gg, N=100, mask = 10, alg = alg, type = 2)
clob<-getRobustness(gg, alg = alg, conmat)
clob
```

---

law-class	<i>Result of PowerLaw fit</i>
-----------	-------------------------------

---

**Description**

Result of PowerLaw 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	<i>Calculate layout based upon membership</i>
-----------------	---

---

**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 <a href="#">calcMembership</a>
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	<i>Calculate two-level layout from recluster matrix</i>
-------------------	---

---

**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 <code>calcReclusterMatrix</code> invocation
layout	one of the layout algorithms from <code>layout_</code>

**Value**

Layout in a form of 2D matrix.

**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	<i>Function to build consensus matrix in memory</i>
---------------------	---

---

**Description**

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.

**Usage**

```
makeConsensusMatrix(
  gg,
  N = 500,
  mask = 20,
  alg,
  type,
  reclust = FALSE,
  Cnmax = 10
)
```

**Arguments**

gg	graph to perturb
N	number of perturbation steps
mask	percentage of elements to perturb
alg	clustering alg.
type	edges=>1 or nodes=>2 to mask
reclust	logical to decide whether to invoke reclustering via <a href="#">recluster</a>
Cnmax	maximum size of the cluster in mem that will not be processed if reclustering is invoked

**Value**

consensus matrix of Nvert X Nvert

**Examples**

```
data(karate, package='igraphdata')
alg<-'louvain'
gg<-calcClustering(karate, alg = alg)
conmat<-makeConsensusMatrix(gg, N=100, mask = 10, alg = alg, type = 2)
dim(conmat)
```

---

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  
)
```

**Arguments**

gg	graph object to analyze
alg	clustering algorithm
Nint	number of iterations

**Details**

Used the normalised network modularity value  $Q_m$  based on the previous studies by Parter et al., 2007, Takemoto, 2012, Takemoto, 2013, Takemoto and Borjigin, 2011, which was defined as:  $Q_m = (Q_{real} - Q_{rand}) / (Q_{max} - Q_{rand})$  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**

```
library(synaptome.db)  
cid<-match('Presynaptic', getCompartments())$Name)  
t<-getAllGenes4Compartment(cid)  
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)  
  
nm<-normModularity(gg, alg='louvain',Nint=10)
```

---

permute	<i>Randomly shuffle annotations</i>
---------	-------------------------------------

---

### Description

This function is a convenience 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	<i>Plot Bridgeness values</i>
----------------	-------------------------------

---

### 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 categorise 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 ( $0 < SI < 0.5$ ;  $0.5 < Br < 1$ )).
- Region 2, proteins having 'global' and 'local' influence ( $0.5 < SI < 1$ ,  $0.5 < Br < 1$ ).
- Region 3, proteins centred within the community they belong to, but also communicating with a few other specific communities ( $0 < SI < 0.5$ ;  $0.1 < Br < 0.5$ ).
- Region 4, proteins with 'local' impact, primarily within one or two communities (local or party hubs,  $0.5 < SI < 1$ ,  $0 < Br < 0.5$ ).



**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 bridgeness values stored as attributes, after call to <a href="#">calcBridgeness</a>
alg	clustering algorithm that was used to calculate bridgeness values
VIPs	list of 'special' 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 'special' 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 'special' genes

**Value**

[ggplot](#) object with plot

**Examples**

```

data(karate, package='igraphdata')
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	<i>Plot graph entropy values versus vertex degree for each perturbed vertex value.</i>
-------------	--

---

**Description**

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

**Usage**

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

**Arguments**

SRprime	results of <a href="#">getEntropy</a> invocation
subTIT	entropy axis label
SRo	second element of results of <a href="#">getEntropyRate</a> invocation
maxSr	first element of results of <a href="#">getEntropyRate</a> invocation

**Value**

ggplot2 object with diagram

**Examples**

```

library(synaptome.db)
cid<-match('Presynaptic',getCompartments())$Name)
t<-getAllGenes4Compartment(cid)
gg<-buildFromSynaptomeByEntrez(t$HumanEntrez)
gg<-annotateGeneNames(gg)
ent <- getEntropyRate(gg)
SRprime <- getEntropy(gg, maxSr = NULL)
plotEntropy(SRprime, subTIT = "Entropy", SRo = ent$SRo, maxSr = ent$maxSr)

```

---

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, GOC-CID, 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)
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 et al. (2013) [doi:10.1016/j.neuron.2013.02.027](https://doi.org/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)
```

**Arguments**

GG graph to cluster

ALGN algorithm to apply

CnMAX maximum size of the cluster in mem that will not be processed

**Value**

membership 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	<i>Remove vertex property.</i>
------------------	--------------------------------

---

**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')
vertex_attr_names(karate)
m<-removeVertexTerm(karate, 'color')
vertex_attr_names(m)
```

---

runPermDisease	<i>Calculate disease-disease pair overlaps on permuted network to estimate its statistical significance</i>
----------------	---

---

### Description

Function to calculate the disease-pair overlap characteristics of an inputted network, before applying  $N_{\text{perm}}$  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

<code>gg</code>	interactome network as igraph object
<code>name</code>	name of the attribute that stores disease annotation
<code>diseases</code>	list of diseases to match
<code>Nperm</code>	number of permutations to apply
<code>permute</code>	type of permutations. random – annotation is randomly shuffled, binned – annotation is shuffled in a way to preserve node degree-annotation relationship by <a href="#">degreeBinnedGDAs</a> .
<code>alpha</code>	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 statistics 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)
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	<i>Function to randomly shuffle vertex annotation terms, whilst preserving the vertex degree originally found with that annotation term..</i>
--------------------	---

---

**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 <a href="#">degreeBinnedGDAs</a>
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)
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, reclust = FALSE, Cnmax = 10)
```

### Arguments

gg	graph
mask	percentage of elements to perturbe
alg	clustering alg.
type	edges=>1 or nodes=>2 to mask
reclust	logical to decide whether to invoke reclustering via <a href="#">recluster</a>
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)
smp1<-BioNAR:::sampleGraphClust(karate, mask=10, alg, type=2)
```



---

SCH_flatfile.csv	<i>Schizophrenia related synaptic gene functional annotation.</i>
------------------	---

---

**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](#)

---

unescapeAnnotation	<i>Unescape annotation strings</i>
--------------------	------------------------------------

---

**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

- \* **diseasome**
  - diseasome.rda, [28](#)
- \* **file**
  - diseasome.rda, [28](#)
  - flatfile.go.BP.csv, [32](#)
  - flatfile.go.CC.csv, [32](#)
  - flatfile.go.MF.csv, [32](#)
  - flatfile\_human\_gene2HDO.csv, [33](#)
  - PPI\_Presynaptic.csv, [51](#)
  - PPI\_Presynaptic.gml, [51](#)
  - PresynAn.csv, [52](#)
  - SCH\_flatfile.csv, [57](#)
- \* **graphs**
  - diseasome.rda, [28](#)
- addEdgeAtts, [3](#)
- annotateGeneNames, [4](#)
- annotateGoBP, [5](#), [32](#)
- annotateGoCC, [6](#), [32](#)
- annotateGoMF, [6](#), [32](#)
- annotateGOont, [7](#)
- annotateInterpro, [8](#)
- annotatePresynaptic, [9](#), [52](#)
- annotateSCHanno, [9](#), [57](#)
- annotateTopOntoOVG, [10](#), [33](#)
- annotateVertex, [11](#), [12](#)
- applpMatrixToGraph, [12](#), [17](#)
- BioNAR, [13](#)
- BioNAR-package (BioNAR), [13](#)
- buildConsensusMatrix, [13](#)
- buildFromSynaptomeByEntrez, [14](#)
- buildFromSynaptomeGeneTable, [14](#)
- buildNetwork, [15](#), [51](#)
- calcAllClustering, [16](#)
- calcBridgeness, [16](#), [49](#)
- calcCentrality, [17](#)
- calcCentralityExternalDistances, [18](#)
- calcCentralityInternalDistances, [19](#)
- calcClustering, [16](#), [20](#)
- calcDiseasePairs, [21](#)
- calcEntropy, [22](#)
- calcMembership, [23](#), [44](#)
- calcReclusterMatrix, [24](#), [45](#)
- calcSparsness, [24](#)
- cluster\_infomap, [37](#)
- cluster\_louvain, [37](#)
- clusteringSummary, [25](#)
- clusterORA, [26](#)
- communities, [37](#)
- degreeBinnedGDAs, [21](#), [27](#), [54](#), [55](#)
- diseasome.rda, [28](#)
- dist, [19](#)
- escapeAnnotation, [28](#)
- evalCentralitySignificance, [29](#)
- fastgreedy.community, [37](#)
- findLCC, [30](#)
- fitDegree, [31](#), [42](#)
- flatfile.go.BP.csv, [32](#)
- flatfile.go.CC.csv, [32](#)
- flatfile.go.MF.csv, [32](#)
- flatfile\_human\_gene2HDO.csv, [33](#)
- getAnnotationList, [33](#)
- getAnnotationVertexList, [34](#)
- getBridgeness, [16](#), [35](#)
- getCentralityMatrix, [17](#), [18](#), [36](#), [41](#), [42](#)
- getClustering, [20](#), [23](#), [37](#)
- getClusterSubgraphByID, [38](#)
- getCommunityGraph, [38](#)
- getDiseases, [39](#)
- getDType, [39](#)
- getEntropy, [40](#), [50](#)
- getEntropyRate, [41](#), [50](#)
- getGenesByID, [15](#)
- getGraphCentralityECDF, [41](#)

getRandomGraphCentrality, [18](#), [19](#), [42](#)  
getRobustness, [43](#)  
ggplot, [49](#)  
graph, [28](#)  
graph.data.frame, [15](#)

keys, [4](#), [8](#)  
keytypes, [4](#), [8](#)

law-class, [44](#)  
layout\_, [45](#)  
layoutByCluster, [44](#)  
layoutByRecluster, [45](#)  
leading.eigenvector.community, [37](#)

makeConsensusMatrix, [45](#)  
match.arg, [20](#), [37](#)  
modularity, [25](#)

normModularity, [46](#)

org.Hs.eg.db, [4](#), [7](#)

permute, [48](#)  
plotBridgeness, [48](#)  
plotEntropy, [50](#)  
PPI\_Presynaptic.csv, [51](#)  
PPI\_Presynaptic.gml, [51](#)  
prepareGDA, [27](#), [51](#)  
PresynAn.csv, [52](#)

recluster, [46](#), [52](#), [56](#)  
removeVertexTerm, [53](#)  
runPermDisease, [54](#)

sample, [48](#)  
sample\_correlated\_gnp, [42](#)  
sample\_gnp, [42](#)  
sample\_pa, [42](#)  
sampleDegBinnedGDA, [55](#)  
sampleGraphClust, [14](#), [56](#)  
SCH\_flatfile.csv, [57](#)  
spectral\_igraph\_communities, [37](#)  
spinglass.community, [37](#)  
synaptome.db, [14](#)

unescapeAnnotation, [57](#)

walktrap.community, [37](#)

zeroNA, [58](#)