Package ‘graphsim’

May 13, 2025

Type Package

Title Simulate Expression Data from 'igraph' Networks

Version 1.0.4

Date 2025-05-02

Description Functions to develop simulated continuous data (e.g., gene expression) from a sigma covariance matrix derived from a graph structure in 'igraph' objects. Intended to extend 'mvtnorm' to take 'igraph' structures rather than sigma matrices as input. This allows the use of simulated data that correctly accounts for pathway relationships and correlations. This allows the use of simulated data that correctly accounts for pathway relationships and correlations. Here we present a versatile statistical framework to simulate correlated gene expression data from biological pathways, by sampling from a multivariate normal distribution derived from a graph structure. This package allows the simulation of biological pathways from a graph structure based on a statistical model of gene expression. For example methods to infer biological pathways and gene regulatory networks from gene expression data can be tested on simulated datasets using this framework. This also allows for pathway structures to be considered as a confounding variable when simulating gene expression data to test the performance of genomic analyses.

License GPL-3

URL https://github.com/TomKellyGenetics/graphsim/

BugReports https://github.com/TomKellyGenetics/graphsim/issues/

Depends R (>= 2.10)

Imports gplots, igraph, mvtnorm, matrixcalc, Matrix, graphics

Suggests devtools, knitr (>= 1.5), markdown, prettydoc, R.rsp, rmarkdown, testthat, scales, vdiffr

LazyData TRUE

RoxygenNote 7.3.2

Encoding UTF-8

VignetteBuilder R.rsp

NeedsCompilation no

Author S. Thomas Kelly [aut, cre],
Michael A. Black [aut, ths],
Robrecht Cannoodt [ctb],
Jason Cory Brunson [ctb]

Maintainer S. Thomas Kelly <`tomkellygenetics@gmail.com`>

Repository CRAN

Date/Publication 2025-05-12 22:10:02 UTC

Contents

`graphsim-package` *The graphsim package*

Description

graphsim is a package to simulate normalised expression data from networks for biological pathways using ‘`igraph`’ objects and multivariate normal distributions.

Details

This package provides functions to develop simulated continuous data (e.g., gene expression) from a Sigma (Σ) covariance matrix derived from a graph structure in ‘`igraph`’ objects. Intended to extend ‘`mvtnorm`’ to take 'igraph' structures rather than sigma matrices as input. This allows the use of simulated data that correctly accounts for pathway relationships and correlations. Here we present a versatile statistical framework to simulate correlated gene expression data from biological pathways, by sampling from a multivariate normal distribution derived from a graph structure. This package allows the simulation of biologicalpathways from a graph structure based on a statistical model of gene expression, such as simulation of expression profiles that of log-transformed and normalised data from microarray and RNA-Seq data.

Introduction

This package enables the generation of simulated gene expression datasets containing pathway relationships from a known underlying network. These simulated datasets can be used to evaluate various bioinformatics methodologies, including statistical and network inference procedures.

These are computed by 1) resolving inhibitory states to derive a consistent matrix of positive and negative edges, 2) inferring relationships between nodes from paths in the graph, 3) weighting these in a Sigma (Σ) covariance matrix and 4) using this to sample a multivariate normal distribution.

Getting Started

The `generate_expression` function is a wrapper around all necessary functions to give a final simulated dataset.

Here we set up an example graph object using the `igraph` package.

```
library("igraph")
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"),c("D", "E"),
                             c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
```

Then we can call `generate_expression` to return the simulated data based on the relationships defined in the graph structure. Various options are available to fine-tune this.

```
expr <- generate_expression(100, graph_structure,
                            cor = 0.8,
                            mean = 0,
                            sd = 1,
                            comm = FALSE,
                            dist = TRUE,
                            absolute = FALSE,
                            laplacian = FALSE)
```

Here we can see the final result. The graph structure defines the covariance matrix used by `rmvnorm` to generate a multivariate distribution.

```
dim(expr)

library("gplots")
heatmap.2(expr,
          scale = "none",
          trace = "none",
          col = bluered(50),
          colsep = 1:4,
          rowsep = 1:4)
```

This dataset consists of 9 rows (one for each vertex or gene) in the graph and 100 columns (one for each sample or observation).

Input with an adjacency matrix is available using the `generate_expression_mat` function.

Creating Input Data

Graph structures can be passed directly from the igraph package. Using this package, you can create an 'igraph' class object.

```
> class(graph_structure)
[1] "igraph"

> graph_structure
IGRAPH ba7fa2f DN-- 9 8 --
  + attr: name (v/c)
  + edges from ba7fa2f (vertex names):
    [1] A->C B->C C->D D->E D->F F->G F->I H->I
```

This 'igraph' object class can be passed directly to generate_expression shown above and internal functions described below: make_sigma_mat_graph, make_sigma_mat_dist_graph, make_distance_graph, and make_state_matrix.

The 'graphsim' package also supports various matrix formats and has functions to handle these. The following functions will compute matrices from an 'igraph' object class:

- make_adjmatrix_graph to derive the adjacency matrix for a graph structure.
- make_commonlink_graph to derive the 'common link' matrix for a graph structure of mutually shared neighbours.
- make_laplacian_graph to derive the Laplacian matrix for a graph structure.

The following functions will compute matrices from an adjacency matrix:

- make_commonlink_adjmat to derive the 'common link' matrix for a graph structure of mutually shared neighbours.
- make_laplacian_adjmat to derive the Laplacian matrix for a graph structure.

We provide some pre-generate pathways from Reactoem database for testing and demonstrations:

- RAF_MAP_graph for the interactions in the "RAF/MAP kinase" cascade (17 vertices and 121 edges).
- Pi3K_graph for the phosphoinositide-3-kinase cascade (35 vertices and 251 edges).
- Pi3K_AKT_graph for the phosphoinositide-3-kinase activation of Protein kinase B pathway "PI3K/AKT activation" (275 vertices and 21106 edges).
- TGFBeta_Smad_graph for the TGF-β receptor signaling activates SMADs pathway (32 vertices and 173 edges).

Please note that demonstrations on larger graph objects. These can be called directly from the pakage:

```
> graphsim::Pi3K_graph
IGRAPH 21437e3 DN-- 35 251 --
  + attr: name (v/c)
  + edges from 21437e3 (vertex names):
```

```
  [1] AKT1->AKT2  AKT1->AKT3  AKT1->CASP9 AKT1->CASP9
  [5] AKT1->CASP9 AKT1->FOXO1 AKT1->FOXO1 AKT1->FOXO1
  [9] AKT1->FOXO3 AKT1->FOXO3 AKT1->FOXO3 AKT1->FOXO4
  [13] AKT1->FOXO4 AKT1->FOXO4 AKT1->GSK3B AKT1->GSK3B
  [17] AKT1->GSK3B AKT1->NOS1  AKT1->NOS2  AKT1->NOS3
  [21] AKT1->PDPK1 AKT2->AKT3  AKT2->CASP9 AKT2->CASP9
  [25] AKT2->CASP9 AKT2->FOXO1 AKT2->FOXO1 AKT2->FOXO1
  [29] AKT2->FOXO3 AKT2->FOXO3 AKT2->FOXO3 AKT2->FOXO4
  + ... omitted several edges
  + ... omitted several edges
```

They can also be imported into R:

```
data(Pi3K_graph)
```

You can assign them to your local environment by calling with from the package:

```
graph_object <- identity(Pi3K_graph)
```

You can also change the object class directly from the package:

```
library("igraph")
Pi3K_adjmat <- as_adjacency_matrix(Pi3K_graph)
```

Pi3K_AKT_graph and TGFBeta_Smad_graph contain graph edge attributes for the 'state' parameter described below.

```
 > TGFBeta_Smad_graph
 IGRAPH f3eac04 DN-- 32 173 --
   + attr: name (v/c), state (e/n)
   + edges from f3eac04 (vertex names):
     [1] BAMBI ->SMAD7  BAMBI ->TGFB1  BAMBI ->TGFBR1 BAMBI ->TGFBR2
     [5] CBL   ->NEDD8  CBL   ->NEDD8  CBL   ->TGFBR2 CBL   ->TGFBR2
     [9] CBL   ->UBE2M  CBL   ->UBE2M  FKBP1A->TGFB1  FKBP1A->TGFBR1
     [13] FKBP1A->TGFBR2 FURIN ->TGFB1  FURIN ->TGFB1  MTMR4 ->SMAD2
     [17] MTMR4 ->SMAD2  MTMR4 ->SMAD3  MTMR4 ->SMAD3  NEDD4L->RPS27A
     [21] NEDD4L->SMAD7  NEDD4L->SMURF1 NEDD4L->SMURF2 NEDD4L->TGFB1
     [25] NEDD4L->TGFBR1 NEDD4L->TGFBR2 NEDD4L->UBA52  NEDD4L->UBB
     [29] NEDD4L->UBC    NEDD8 ->TGFBR2 NEDD8 ->UBE2M  PMEPA1->SMAD2
     + ... omitted several edges

 > E(TGFBeta_Smad_graph)$state
 [1] 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [32] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [63] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [94] 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [125] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [156] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1
```

```
> states <- E(TGFBeta_Smad_graph)$state
> table(states)
states
1   2
103  70
```

Internal Functions

The following functions are used by generate_expression to compute a simulated dataset. They can be called separately to summarise the steps used to compute the final data matrix or for troubleshooting.

- make_sigma_mat_adjmat, make_sigma_mat_comm, make_sigma_mat_laplacian, and make_sigma_mat_graph will compute a Sigma (Σ) covariance matrix from an adjacency matrix, common link matrix, Laplacian matrix, or an 'igraph' object. There are computed as above and passed to rmvnorm.
- make_distance_adjmat, make_distance_comm, make_distance_laplacian, and make_distance_graph will compute a distance matrix of relationships from an adjacency matrix, common link matrix, Laplacian matrix, or an 'igraph' object. There are computed as above and passed to make_sigma.
- make_state_matrix will compute a "state matrix" resolving positive and negative correlations from a vector of edge properties. This is called by make_sigma and generate_expression to ensure that the signs of correlations are consistent.

Examining Step-by-Step

These internal functions can be called to compute steps of the simulation procedure and examine the results.

1. first we create a graph structure and define the input parameters

```
library("igraph")
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"),c("D", "E"),
                             c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
#sample size
data.n <- 100
#data distributions
data.cor <- 0.75
data.mean <- 3
data.sd <- 1.5
#inhibition states
edge_states <- c(1, 1, -1, -1, 1, 1, 1, 1)
```

2. examine the relationships between the genes.

Here we can see which nodes share an edge:

```
> adjacency_matrix <- make_adjmatrix_graph(graph_structure)
> adjacency_matrix
  A C B D E F G I H
A 0 1 0 0 0 0 0 0 0
C 1 0 1 1 0 0 0 0 0
B 0 1 0 0 0 0 0 0 0
D 0 1 0 0 1 1 0 0 0
E 0 0 0 1 0 0 0 0 0
F 0 0 0 1 0 0 1 1 0
G 0 0 0 0 0 1 0 0 0
I 0 0 0 0 0 1 0 0 1
H 0 0 0 0 0 0 0 1 0
```

Here we define a geometrically decreasing series of relationships between genes based on distance by paths in the graph:

```
> relationship_matrix <- make_distance_graph(graph_structure, absolute = FALSE)
> relationship_matrix
  A          C          B          D          E          F          G          I          H
A 1.00000000 0.20000000 0.10000000 0.10000000 0.06666667 0.06666667 0.05000000 0.05000000 0.04000000
C 0.20000000 1.00000000 0.20000000 0.20000000 0.10000000 0.10000000 0.06666667 0.06666667 0.05000000
B 0.10000000 0.20000000 1.00000000 0.10000000 0.06666667 0.06666667 0.05000000 0.05000000 0.04000000
D 0.10000000 0.20000000 0.10000000 1.00000000 0.20000000 0.20000000 0.10000000 0.10000000 0.06666667
E 0.06666667 0.10000000 0.06666667 0.20000000 1.00000000 0.10000000 0.06666667 0.06666667 0.05000000
F 0.06666667 0.10000000 0.06666667 0.20000000 0.10000000 1.00000000 0.20000000 0.20000000 0.10000000
G 0.05000000 0.06666667 0.05000000 0.10000000 0.06666667 0.20000000 1.00000000 0.10000000 0.06666667
I 0.05000000 0.06666667 0.05000000 0.10000000 0.06666667 0.20000000 0.10000000 1.00000000 0.20000000
H 0.04000000 0.05000000 0.04000000 0.06666667 0.05000000 0.10000000 0.06666667 0.20000000 1.00000000
```

Here we can see the resolved edge states through paths in the adjacency matrix:

```
> names(edge_states) <- apply(graph_structure_edges, 1, paste, collapse = "-")
> edge_states
A-C B-C C-D D-E D-F F-G F-I H-I
1   1  -1  -1   1   1   1   1
> state_matrix <- make_state_matrix(graph_structure, state = edge_states)
> state_matrix
   A  C  B  D  E  F  G  I  H
A  1  1  1 -1  1 -1 -1 -1 -1
C  1  1  1 -1  1 -1 -1 -1 -1
B  1  1  1 -1  1 -1 -1 -1 -1
D -1 -1 -1  1 -1  1  1  1  1
E  1  1  1 -1  1 -1 -1 -1 -1
F -1 -1 -1  1 -1  1  1  1  1
G -1 -1 -1  1 -1  1  1  1  1
I -1 -1 -1  1 -1  1  1  1  1
H -1 -1 -1  1 -1  1  1  1  1
```

3. define a Sigma (Σ) covariance matrix

Here we can see that the signs match the `state_matrix` and the covariance is based on the `relationship_matrix` weighted by the correlation (`cor`) and standard deviation (`sd`) parameters.

Note that where `sd` = 1, the diagonals will be 1 and the off-diagonal terms will be correlations.

```
> sigma_matrix <- make_sigma_mat_dist_graph(
+     graph_structure,
+     state = edge_states,
+     cor = data.cor,
+     sd = data.sd,
+     absolute = FALSE
+ )
> sigma_matrix
   A         C         B         D        E         F        G         I         H
A  2.250000  1.687500  0.843750 -0.84375  0.562500 -0.56250 -0.421875 -0.421875 -0.337500
C  1.687500  2.250000  1.687500 -1.68750  0.843750 -0.84375 -0.562500 -0.562500 -0.421875
B  0.843750  1.687500  2.250000 -0.84375  0.562500 -0.56250 -0.421875 -0.421875 -0.337500
D -0.843750 -1.687500 -0.843750  2.25000 -1.687500  1.68750  0.843750  0.843750  0.562500
E  0.562500  0.843750  0.562500 -1.68750  2.250000 -0.84375 -0.562500 -0.562500 -0.421875
F -0.562500 -0.843750 -0.562500  1.68750 -0.843750  2.25000  1.687500  1.687500  0.843750
G -0.421875 -0.562500 -0.421875  0.84375 -0.562500  1.68750  2.250000  0.843750  0.562500
I -0.421875 -0.562500 -0.421875  0.84375 -0.562500  1.68750  0.843750  2.250000  1.687500
H -0.337500 -0.421875 -0.337500  0.56250 -0.421875  0.84375  0.562500  1.687500  2.250000
```

4. generate an expression dataset using this sigma matrix

We use `generate_expression` to compute and expression dataset, simulated using these parameters:

```
> expression_data <- generate_expression(
+     n = data.n,
+     graph_structure,
+     state = edge_states,
+     cor = data.cor,
+     mean = data.mean,
+     sd = data.sd,
+     comm = FALSE,
+     dist = FALSE,
+     absolute = FALSE,
+     laplacian = FALSE
+ )
> dim(expression_data)
[1]   9 100
```

Here we also compute the final observed correlations in the simulated dataset:

```
> cor_data <- cor(t(expression_data))
> dim(cor_data)
[1] 9 9
```

These functions are demonstrated in more detail in the main vignette.

Data Visualization

Heatmaps can be used from the `gplots` package to display these simulated datasets.

```
library("gplots")
heatmap.2(adjacency_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "white", "black"), key = FALSE)

heatmap.2(relationship_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))

heatmap.2(state_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "royalblue", "palevioletred"),
          colsep = 1:length(V(graph_structure)),
          rowsep = 1:length(V(graph_structure)))

heatmap.2(sigma_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "royalblue", "white", "palevioletred"),
          colsep = 1:length(V(graph_structure)),
          rowsep = 1:length(V(graph_structure)))

heatmap.2(expression_data, scale = "none", trace = "none",
          col = colorpanel(50, "royalblue", "white", "palevioletred"),
          colsep = 1:length(V(graph_structure)),
         rowsep = 1:length(V(graph_structure)))

heatmap.2(cor_data, scale = "none", trace = "none",
          col = colorpanel(50, "royalblue", "white", "palevioletred"),
          colsep = 1:length(V(graph_structure)),
          rowsep = 1:length(V(graph_structure)))
```

In particular we can see here that the expected correlations show by the `sigma_matrix` are similar to the observed correlations in the `cor_data`.

Graph Visualization

The ‘graphsim’ package comes with a built-in plotting function to display graph objects.

```
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"),c("D", "E"),
                             c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
plot_directed(graph_structure, layout = layout.kamada.kawai)
```

This supports the ‘state’ parameter to display activating relationships (with positive correlations) and inhibiting or repressive relationships (with negative correlations).

```
edge_states <- c(1, 1, -1, -1, 1, -1, 1, -1)
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
plot_directed(graph_structure, state = edge_states,
              col.arrow = c("darkgreen", "red")[edge_states / 2 + 1.5]
              layout = layout.kamada.kawai)
```

These states can also be passed from the ‘state’ edge attribute of the graph object.

```
graph_pathway <- identity(TGFBeta_Smad_graph)
edge_properties <- E(graph_pathway)$state
plot_directed(graph_pathway,
                col.arrow = c(alpha("navyblue", 0.25),
                                alpha("red", 0.25))[edge_properties],
                fill.node = c("lightblue"),
                layout = layout.kamada.kawai)
```

This plotting function is demonstrated in more detail in the plots_directed.Rmd plotting vignette.

Further information

The graphsim package is published in the *Journal of Open Source Software*. See the paper here for more details: doi:10.21105/joss.02161

The graphsim GitHub repository is here: TomKellyGenetics/graphsim You can find the development version and submit an issue if you have questions or comments.

Citation

To cite package ’graphsim’ in publications use:

Kelly, S.T. and Black, M.A. (2020). graphsim: An R package for simulating gene expression data from graph structures of biological pathways. *Journal of Open Source Software*, **5**(51), 2161, doi:10.21105/joss.02161

A BibTeX entry for LaTeX users is:

```
  @article{Kelly2020joss02161,
     doi = {10.21105/joss.02161},
     year = {2020},
     publisher = {The Open Journal},
     volume = {5},
     number = {51},
     pages = {2161},
     author = {S. Thomas Kelly and Michael A. Black},
    title = {graphsim: An R package for simulating gene expression data from graph structures of biologic
     journal = {Journal of Open Source Software}
    }
```

Author(s)

Maintainer: Tom Kelly <tom.kelly@riken.jp>

Authors:

- Tom Kelly (RIKEN IMS) ORCID)
- Mik Black (Otago University) (ORCID)

Reviewers:

- Cory Brunson (UConn) (ORCID)
- Robrecht Cannoodt (Ghent University) (ORCID)

Editor: Mark Jensen (Frederick National Laboratory for Cancer Research)

See Also

Publication at *Journal of Open Source Software*:

- doi:10.21105/joss.02161

GitHub repository:

- `https://github.com/TomKellyGenetics/graphsim/`

Report bugs:

- `https://github.com/TomKellyGenetics/graphsim/issues`

Contributions:

- `https://github.com/TomKellyGenetics/graphsim/blob/master/CONTRIBUTING.md`

`generate_expression` *Generate Simulated Expression*

Description

Compute simulated continuous expression data from a graph network structure. Requires an `igraph` pathway structure and a matrix of states (1 for activating and -1 for inhibiting) for link signed correlations, from a vector of edge states to a signed adjacency matrix for use in `generate_expression`. Uses graph structure to pass a sigma covariance matrix from `make_sigma_mat_graph` or `make_sigma_mat_dist_graph` on to `rmvnorm`. By default data is generated with a mean of 0 and standard deviation of 1 for each gene (with correlations between derived from the graph structure).

Usage

```
generate_expression(
  n,
  graph,
  state = NULL,
  cor = 0.8,
  mean = 0,
  sd = 1,
  comm = FALSE,
  dist = FALSE,
  absolute = FALSE,
  laplacian = FALSE
)

generate_expression_mat(
  n,
  mat,
  state = NULL,
  cor = 0.8,
  mean = 0,
  sd = 1,
  comm = FALSE,
  dist = FALSE,
  absolute = FALSE,
  laplacian = FALSE
)
```

Arguments

`n`	number of observations (simulated samples).
`graph`	An `igraph` object. May must be directed if states are used.
`state`	numeric vector. Vector of length E(graph). Sign used to calculate state matrix, may be an integer state or inferred directly from expected correlations for each edge. May be applied a scalar across all edges or as a vector for each edge respectively. May also be entered as text for "activating" or "inhibiting" or as integers for activating (0,1) or inhibiting (-1,2). Compatible with inputs for `plot_directed`. Also takes a pre-computed state matrix from `make_state` if applied to the same graph multiple times.
`cor`	numeric. Simulated maximum correlation/covariance of two adjacent nodes. Default to 0.8.
`mean`	mean value of each simulated gene. Defaults to 0. May be entered as a scalar applying to all genes or a vector with a separate value for each.
`sd`	standard deviations of each gene. Defaults to 1. May be entered as a scalar applying to all genes or a vector with a separate value for each.
`comm`, `absolute`, `laplacian`	logical. Parameters for Sigma matrix generation. Passed on to `make_sigma` or `make_sigma`.

dist	logical. Whether a graph distance make_sigma_mat_graph or derived matrix make_sigma_mat_dist_graph is used to compute the sigma matrix (using make_distance).
mat	precomputed adjacency, laplacian, commonlink, or scaled distance matrix (generated by make_distance).

Value

numeric matrix of simulated data (log-normalised counts)

Author(s)

Tom Kelly <tom.kelly@riken.jp>

See Also

See also make_sigma for computing the Sigma (Σ) matrix, make_distance for computing distance from a graph object, and make_state for resolving inhibiting states.

See also plot_directed for plotting graphs or heatmap.2 for plotting matrices.

See also make_laplacian, make_commonlink, or make_adjmatrix for computing input matrices.

See also igraph for handling graph objects.

Other graphsim functions: make_adjmatrix, make_commonlink, make_distance, make_laplacian, make_sigma, make_state, plot_directed()

Other generate simulated expression functions: make_distance, make_sigma, make_state

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)

# compute a simulated dataset for toy example
# n = 100 samples
# cor = 0.8 max correlation between samples
# absolute = FALSE (geometric distance by default)
test_data <- generate_expression(100, graph_test, cor = 0.8)
##' # visualise matrix
library("gplots")
# expression data
heatmap.2(test_data, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# correlations
heatmap.2(cor(t(test_data)), scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
# expected correlations (\eqn{\Sigma})
sigma_matrix <- make_sigma_mat_graph(graph_test, cor = 0.8)
heatmap.2(make_sigma_mat_graph(graph_test, cor = 0.8),
          scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
```

```
# compute adjacency matrix for toy example
adjacency_matrix <- make_adjmatrix_graph(graph_test)
# generate simulated data from adjacency matrix input
test_data <- generate_expression_mat(100, adjacency_matrix, cor = 0.8)

# compute a simulated dataset for toy example
# n = 100 samples
# cor = 0.8 max correlation between samples
# absolute = TRUE (arithmetic distance)
test_data <- generate_expression(100, graph_test, cor = 0.8, absolute = TRUE)
##' # visualise matrix
library("gplots")
# expression data
heatmap.2(test_data, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# correlations
heatmap.2(cor(t(test_data)),
          scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
# expected correlations (\eqn{\Sigma})
sigma_matrix <- make_sigma_mat_graph(graph_test, cor = 0.8)
heatmap.2(make_sigma_mat_graph(graph_test, cor = 0.8),
          scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)

# compute a simulated dataset for toy network
# n = 250 samples
# state = edge_state (properties of each edge)
# cor = 0.95 max correlation between samples
# absolute = FALSE (geometric distance by default)
edge_state <- c(1, 1, -1, 1, 1, 1, 1, -1)
structure_data <- generate_expression(250, graph_structure,
                                      state = edge_state, cor = 0.95)
##' # visualise matrix
library("gplots")
# expression data
heatmap.2(structure_data, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# correlations
heatmap.2(cor(t(structure_data)), scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# expected correlations (\eqn{\Sigma})
sigma_matrix <- make_sigma_mat_graph(graph_structure,
                                     state = edge_state, cor = 0.8)
heatmap.2(make_sigma_mat_graph(graph_structure,
                               state = edge_state, cor = 0.8),
          scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))

# compute adjacency matrix for toy network
graph_structure_adjacency_matrix <- make_adjmatrix_graph(graph_structure)
# define states for for each edge
edge_state <- c(1, 1, -1, 1, 1, 1, 1, -1)
# generate simulated data from adjacency matrix input
structure_data <- generate_expression_mat(250, graph_structure_adjacency_matrix,
                                          state = edge_state, cor = 0.8)

# compute a simulated dataset for toy network
# n = 1000 samples
# state = TGFBeta_Smad_state (properties of each edge)
# cor = 0.75 max correlation between samples
# absolute = FALSE (geometric distance by default)
 # compute states directly from graph attributes for TGF-\eqn{\Beta} pathway
TGFBeta_Smad_state <- E(TGFBeta_Smad_graph)$state
table(TGFBeta_Smad_state)
# generate simulated data
TGFBeta_Smad_data <- generate_expression(1000, TGFBeta_Smad_graph, cor = 0.75)
##' # visualise matrix
library("gplots")
# expression data
heatmap.2(TGFBeta_Smad_data, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# correlations
heatmap.2(cor(t(TGFBeta_Smad_data)), scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))
# expected correlations (\eqn{\Sigma})
sigma_matrix <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph, cor = 0.75)
heatmap.2(make_sigma_mat_dist_graph(TGFBeta_Smad_graph, cor = 0.75),
          scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))


# generate simulated data (absolute distance and shared edges)
TGFBeta_Smad_data <- generate_expression(1000, TGFBeta_Smad_graph,
                                         cor = 0.75, absolute = TRUE, comm = TRUE)
##' # visualise matrix
library("gplots")
# expression data
heatmap.2(TGFBeta_Smad_data, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
# correlations
heatmap.2(cor(t(TGFBeta_Smad_data)), scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))
# expected correlations (\eqn{\Sigma})
sigma_matrix <- make_sigma_mat_graph(TGFBeta_Smad_graph,
                                     cor = 0.75, comm = TRUE)
heatmap.2(make_sigma_mat_graph(TGFBeta_Smad_graph, cor = 0.75, comm = TRUE),
          scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))
```

---

make_adjmatrix *Generate Adjacency Matrix*

Description

Compute the adjacency matrix of a (directed) igraph structure, preserving node/column/row names (and direction).

Usage

```
make_adjmatrix_graph(graph, directed = FALSE)
```

Arguments

graph	An igraph object. May be directed or weighted.
directed	logical. Whether directed information is passed to the adjacency matrix.

Value

An adjacency matrix compatible with generating an expression matrix

Author(s)

Tom Kelly <tom.kelly@riken.jp>

See Also

See also generate_expression for computing the simulated data, make_sigma for computing the Sigma (Σ) matrix, make_distance for computing distance from a graph object, make_state for resolving inhibiting states.

See also plot_directed for plotting graphs or heatmap.2 for plotting matrices.

See also make_laplacian or make_commonlink for computing input matrices.

See also igraph for handling graph objects.

Other graphsim functions: generate_expression(), make_commonlink, make_distance, make_laplacian, make_sigma, make_state, plot_directed()

Other graph conversion functions: make_commonlink, make_laplacian

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)

# compute adjacency matrix for toy example
adjacency_matrix <- make_adjmatrix_graph(graph_test)
adjacency_matrix

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
# compute adjacency matrix for toy network
graph_structure_adjacency_matrix <- make_adjmatrix_graph(graph_structure)
graph_structure_adjacency_matrix

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)

# compute adjacency matrix for TGF-\eqn{\Beta} receptor signaling activates SMADs
TGFBeta_Smad_adjacency_matrix <- make_adjmatrix_graph(TGFBeta_Smad_graph)
dim(TGFBeta_Smad_adjacency_matrix)
TGFBeta_Smad_adjacency_matrix[1:12, 1:12]
```

---

make_commonlink *Generate Common Link Matrix*

Description

Compute the common link matrix of a (directed) igraph structure, preserving node / column / row names (and direction). We can compute the common links between each pair of nodes. This shows how many nodes are mutually connected to both of the nodes in the matrix (how many paths of length 2 exist between them).

Usage

```
make_commonlink_adjmat(adj_mat)

make_commonlink_graph(graph, directed = FALSE)
```

Arguments

adj_mat	precomputed adjacency matrix.
graph	An igraph object. May be directed or weighted.
directed	logical. Whether directed information is passed to the adjacency matrix.

Value

An integer matrix of number of links shared between nodes

Author(s)

Tom Kelly `<tom.kelly@riken.jp>`

See Also

See also `generate_expression` for computing the simulated data, `make_sigma` for computing the Sigma (Σ) matrix, `make_distance` for computing distance from a graph object, `make_state` for resolving inhibiting states.

See also `plot_directed` for plotting graphs or `heatmap.2` for plotting matrices.

See also `make_laplacian` or `make_adjmatrix` for computing input matrices.

See also `igraph` for handling graph objects.

Other graphsim functions: `generate_expression()`, `make_adjmatrix`, `make_distance`, `make_laplacian`, `make_sigma`, `make_state`, `plot_directed()`

Other graph conversion functions: `make_adjmatrix`, `make_laplacian`

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)

# compute adjacency matrix for toy example
adjacency_matrix <- make_adjmatrix_graph(graph_test)
# compute nodes with shared edges to a 3rd node
common_link_matrix <- make_commonlink_adjmat(adjacency_matrix)
common_link_matrix

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
# compute adjacency matrix for toy network
graph_structure_adjacency_matrix <- make_adjmatrix_graph(graph_structure)
# compute nodes with shared edges to a 3rd node
graph_structure_common_link_matrix <- make_commonlink_adjmat(graph_structure_adjacency_matrix)
graph_structure_common_link_matrix

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)
# compute nodes with shared edges to a 3rd node
TGFBeta_Smad_adjacency_matrix <- make_adjmatrix_graph(TGFBeta_Smad_graph)
TGFBeta_Smad_common_link_matrix <- make_commonlink_adjmat(TGFBeta_Smad_adjacency_matrix)
# we show summary statistics as the graph is large
```

```
dim(TGFBeta_Smad_common_link_matrix)
TGFBeta_Smad_common_link_matrix[1:12, 1:12]
# visualise matrix
library("gplots")
heatmap.2(TGFBeta_Smad_common_link_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
```

`make_distance` *Generate Distance Matrix*

Description

Compute the distance matrix of using shortest paths of a (directed) `igraph` structure, normalising by the diameter of the network, preserving node/column/row names (and direction). This is used to compute the simulatted data for `generate_expression` (when `dist` = TRUE) by `make_sigma_mat_dist_graph`.

Usage

```
make_distance_graph(graph, directed = FALSE, absolute = FALSE)

make_distance_adjmat(mat, directed = FALSE, absolute = FALSE)

make_distance_comm(mat, directed = FALSE, absolute = FALSE)

make_distance_laplacian(mat, directed = FALSE, absolute = FALSE)
```

Arguments

`graph`	An `igraph` object. May be directed or weighted.
`directed`	logical. Whether directed information is passed to the distance matrix.
`absolute`	logical. Whether distances are scaled as the absolute difference from the diameter (maximum possible). Defaults to TRUE. The alternative is to calculate a relative difference from the diameter for a geometric decay in distance.
`mat`	precomputed adjacency or commonlink matrix.

Value

A numeric matrix of values in the range [0, 1] where higher values are closer in the network

Author(s)

Tom Kelly <`tom.kelly@riken.jp`>

See Also

See also generate_expression for computing the simulated data, make_sigma for computing the Sigma (Σ) matrix, make_state for resolving inhibiting states.

See also plot_directed for plotting graphs or heatmap.2 for plotting matrices.

See also make_laplacian, make_commonlink, or make_adjmatrix for computing input matrices.

See also igraph for handling graph objects.

Other graphsim functions: generate_expression(), make_adjmatrix, make_commonlink, make_laplacian, make_sigma, make_state, plot_directed()

Other generate simulated expression functions: generate_expression(), make_sigma, make_state

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)

# compute adjacency matrix for toy example
adjacency_matrix <- make_adjmatrix_graph(graph_test)
# compute nodes with relationships between nodes (geometrically decreasing by default)
distance_matrix_geom <- make_distance_adjmat(adjacency_matrix)
distance_matrix_geom

# compute nodes with relationships between nodes (arithmetically decreasing)
distance_matrix_abs <- make_distance_adjmat(adjacency_matrix, absolute = TRUE)
distance_matrix_abs

# compute Laplacian matrix
laplacian_matrix <- make_laplacian_graph(graph_test)
# compute distances from Laplacian
distance_matrix <- make_distance_laplacian(laplacian_matrix)

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
# compute adjacency matrix for toy network
graph_structure_adjacency_matrix <- make_adjmatrix_graph(graph_structure)
# compute nodes with relationships between nodes (geometrically decreasing by default)
graph_structure_distance_matrix_geom <- make_distance_adjmat(graph_structure_adjacency_matrix)
graph_structure_distance_matrix_geom
# visualise matrix
library("gplots")
heatmap.2(graph_structure_distance_matrix_geom, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
# compute nodes with relationships between nodes (arithmetically decreasing)
graph_structure_distance_matrix_abs <- make_distance_adjmat(graph_structure_adjacency_matrix,
                                                            absolute = TRUE)
graph_structure_distance_matrix_abs
# visualise matrix
```

```
library("gplots")
heatmap.2(graph_structure_distance_matrix_abs,
          scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)
# compute nodes with relationships between nodes (geometrically decreasing by default)
TGFBeta_Smad_adjacency_matrix <- make_adjmatrix_graph(TGFBeta_Smad_graph)
TGFBeta_Smad_distance_matrix_geom <- make_distance_adjmat(TGFBeta_Smad_adjacency_matrix)
# visualise matrix
library("gplots")
heatmap.2(TGFBeta_Smad_distance_matrix_geom, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
# compute nodes with relationships between nodes (arithmetically decreasing)
TGFBeta_Smad_distance_matrix_abs <- make_distance_adjmat(TGFBeta_Smad_adjacency_matrix,
                        absolute = TRUE)
# visualise matrix
library("gplots")
heatmap.2(TGFBeta_Smad_distance_matrix_abs, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
```

`make_laplacian` *Generate Laplacian Matrix*

Description

Compute the Laplacian matrix of a (directed) `igraph` structure, preserving node/column/row names (and direction).

Usage

```
make_laplacian_adjmat(mat, directed = FALSE)

make_laplacian_graph(graph, directed = FALSE)
```

Arguments

`mat`	precomputed adjacency matrix.
`directed`	logical. Whether directed information is passed to the Laplacian matrix.
`graph`	An `igraph` object. May be directed or weighted.

Value

An Laplacian matrix compatible with generating an expression matrix

Author(s)

Tom Kelly <tom.kelly@riken.jp>

See Also

See also generate_expression for computing the simulated data, make_sigma for computing the Sigma (Σ) matrix, make_distance for computing distance from a graph object, make_state for resolving inhibiting states.

See also plot_directed for plotting graphs or heatmap.2 for plotting matrices.

See also make_commonlink or make_adjmatrix for computing input matrices.

See also igraph for handling graph objects.

Other graphsim functions: generate_expression(), make_adjmatrix, make_commonlink, make_distance, make_sigma, make_state, plot_directed()

Other graph conversion functions: make_adjmatrix, make_commonlink

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)
# compute Laplacian matrix for toy example
laplacian_matrix <- make_laplacian_graph(graph_test)
laplacian_matrix

# compute Laplacian matrix from adjacency matrix
adjacency_matrix <- make_adjmatrix_graph(graph_test)
laplacian_matrix <- make_laplacian_adjmat(adjacency_matrix)
laplacian_matrix

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)
# compute Laplacian matrix for toy network
graph_structure_laplacian_matrix <- make_laplacian_graph(graph_structure)
graph_structure_laplacian_matrix

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)

# compute Laplacian matrix for TGF-\eqn{\Beta} receptor signaling activates SMADs
TGFBeta_Smad_laplacian_matrix <- make_laplacian_graph(TGFBeta_Smad_graph)
dim(TGFBeta_Smad_laplacian_matrix)
TGFBeta_Smad_laplacian_matrix[1:12, 1:12]
# visualise matrix
library("gplots")
heatmap.2(TGFBeta_Smad_laplacian_matrix, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
```

make_sigma *Generate Sigma (Σ) Matrix*

Description

Compute the Sigma (Σ) matrix from an igraph structure or pre-computed matrix. These are compatible with rmvnorm and generate_expression. By default data is generated with a mean of 0 and standard deviation of 1 for each gene (with correlations between derived from the graph structure). Thus where the Sigma (Σ) matrix has diagonals of 1 (for the variance of each gene) then the symmetric non-diagonal terms (for covariance) determine the correlations between each gene in the output from generate_expression.

Usage

```
make_sigma_mat_adjmat(mat, state = NULL, cor = 0.8, sd = 1)

make_sigma_mat_comm(mat, state = NULL, cor = 0.8, sd = 1)

make_sigma_mat_laplacian(mat, state = NULL, cor = 0.8, sd = 1)

make_sigma_mat_graph(
  graph,
  state = NULL,
  cor = 0.8,
  sd = 1,
  comm = FALSE,
  laplacian = FALSE,
  directed = FALSE
)

make_sigma_mat_dist_adjmat(
  mat,
  state = NULL,
  cor = 0.8,
  sd = 1,
  absolute = FALSE
)

make_sigma_mat_dist_graph(
  graph,
  state = NULL,
  cor = 0.8,
  sd = 1,
  absolute = FALSE
)
```

Arguments

`mat`	precomputed adjacency, laplacian, commonlink, or scaled distance matrix (generated by `make_distance`).
`state`	numeric vector. Vector of length E(graph). Sign used to calculate state matrix, may be an integer state or inferred directly from expected correlations for each edge. May be applied a scalar across all edges or as a vector for each edge respectively. May also be entered as text for "activating" or "inhibiting" or as integers for activating (0,1) or inhibiting (-1,2). Compatible with inputs for `plot_directed`. Also takes a pre-computed state matrix from `make_state` if applied to the same graph multiple times.
`cor`	numeric. Simulated maximum correlation/covariance of two adjacent nodes. Default to 0.8.
`sd`	standard deviations of each gene. Defaults to 1. May be entered as a scalar applying to all genes or a vector with a separate value for each.
`graph`	An `igraph` object. May be directed or weighted.
`comm`	logical whether a common link matrix is used to compute sigma. Defaults to FALSE (adjacency matrix).
`laplacian`	logical whether a Laplacian matrix is used to compute sigma. Defaults to FALSE (adjacency matrix).
`directed`	logical. Whether directed information is passed to the distance matrix.
`absolute`	logical. Whether distances are scaled as the absolute difference from the diameter (maximum possible). Defaults to TRUE. The alternative is to calculate a relative difference from the diameter for a geometric decay in distance.

Value

a numeric covariance matrix of values in the range [-1, 1]

Author(s)

Tom Kelly `<tom.kelly@riken.jp>`

See Also

See also `generate_expression` for computing the simulated data, `make_distance` for computing distance from a graph object, and `make_state` for resolving inhibiting states.

See also `plot_directed` for plotting graphs or `heatmap.2` for plotting matrices.

See also `make_laplacian`, `make_commonlink`, or `make_adjmatrix` for computing input matrices.

See also `igraph` for handling graph objects.

Other graphsim functions: `generate_expression()`, `make_adjmatrix`, `make_commonlink`, `make_distance`, `make_laplacian`, `make_state`, `plot_directed()`

Other generate simulated expression functions: `generate_expression()`, `make_distance`, `make_state`

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)
# compute sigma (\eqn{\Sigma}) matrix for toy example
sigma_matrix <- make_sigma_mat_graph(graph_test, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix  from adjacency matrix for toy example
adjacency_matrix <- make_adjmatrix_graph(graph_test)
sigma_matrix <- make_sigma_mat_adjmat(adjacency_matrix, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from shared edges for toy example
common_link_matrix <- make_commonlink_graph(graph_test)
sigma_matrix <- make_sigma_mat_comm(common_link_matrix, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from Laplacian for toy example
laplacian_matrix <- make_laplacian_graph(graph_test)
sigma_matrix <- make_sigma_mat_laplacian(laplacian_matrix, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from distance matrix for toy example
distance_matrix <- make_distance_graph(graph_test, absolute = FALSE)
sigma_matrix <- make_sigma_mat_dist_adjmat(distance_matrix, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from toy example graph
sigma_matrix <- make_sigma_mat_dist_graph(graph_test, cor = 0.8)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from absolute distance directly from toy example graph
sigma_matrix <- make_sigma_mat_dist_graph(graph_test, cor = 0.8, absolute = TRUE)
sigma_matrix

# compute sigma (\eqn{\Sigma}) matrix from geometric distance with sd = 2
sigma_matrix <- make_sigma_mat_dist_graph(graph_test, cor = 0.8, sd = 2)
sigma_matrix

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from synthetic graph network
sigma_matrix_graph_structure <- make_sigma_mat_dist_graph(graph_structure,
                                                          cor = 0.8, absolute = FALSE)
sigma_matrix_graph_structure
# visualise matrix
library("gplots")
```

```
heatmap.2(sigma_matrix_graph_structure, scale = "none", trace = "none",
                     col = colorpanel(50, "white", "red"))

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from
# synthetic graph network with inhibitions
edge_state <- c(1, 1, -1, 1, 1, 1, 1, -1)
# pass edge state as a parameter
sigma_matrix_graph_structure_inhib <- make_sigma_mat_dist_graph(graph_structure,
                                                                state = edge_state,
                                                                cor = 0.8,
                                                                absolute = FALSE)
sigma_matrix_graph_structure_inhib
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_graph_structure_inhib, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from
# synthetic graph network with inhibitions
E(graph_structure)$state <-  c(1, 1, -1, 1, 1, 1, 1, -1)
# pass edge state as a graph attribute
sigma_matrix_graph_structure_inhib <- make_sigma_mat_dist_graph(graph_structure,
                                                                cor = 0.8,
                                                                absolute = FALSE)
sigma_matrix_graph_structure_inhib
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_graph_structure_inhib, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from TGF-\eqn{\Beta} pathway
TFGBeta_Smad_state <- E(TGFBeta_Smad_graph)$state
table(TFGBeta_Smad_state)
# states are edge attributes
 sigma_matrix_TFGBeta_Smad_inhib <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph,
                                                              cor = 0.8,
                                                              absolute = FALSE)
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_TFGBeta_Smad_inhib, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from TGF-\eqn{\Beta} pathway
TGFBeta_Smad_graph <- remove.edge.attribute(TGFBeta_Smad_graph, "state")
# compute with states removed (all negative)
sigma_matrix_TFGBeta_Smad <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph,
                                                       state = -1,
                                                       cor = 0.8,
                                                       absolute = FALSE)
```

```
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_TFGBeta_Smad, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))
# compute with states removed (all positive)
sigma_matrix_TFGBeta_Smad <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph,
                                                        state = 1,
                                                        cor = 0.8,
                                                        absolute = FALSE)
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_TFGBeta_Smad, scale = "none", trace = "none",
          col = colorpanel(50, "white", "red"))

#restore edge attributes
TGFBeta_Smad_graph <- set_edge_attr(TGFBeta_Smad_graph, "state",
                                    value = TFGBeta_Smad_state)
TFGBeta_Smad_state <- E(TGFBeta_Smad_graph)$state
# states are edge attributes
 sigma_matrix_TFGBeta_Smad_inhib <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph,
                                                               cor = 0.8,
                                                               absolute = FALSE)
# visualise matrix
library("gplots")
heatmap.2(sigma_matrix_TFGBeta_Smad_inhib, scale = "none", trace = "none",
          col = colorpanel(50, "blue", "white", "red"))
```

make_state *Make State Matrix*

Description

Functions to compute the matrix of states (1 for activating and -1 for inhibiting) for link signed correlations, from a vector of edge states to a signed adjacency matrix for use in generate_expression. This resolves edge states to determine the sign of all correlations between nodes in a network. These are computed interally for sigma matrices as required.

Usage

```
make_state_matrix(graph, state = NULL)
```

Arguments

graph	An igraph object. May be directed or weighted as long as a shortest path can be computed.
state	numeric vector. Vector of length E(graph). Sign used to calculate state matrix, may be an integer state or inferred directly from expected correlations for each edge. May be applied a scalar across all edges or as a vector for each

edge respectively. May also be entered as text for "activating" or "inhibiting" or as integers for activating (0,1) or inhibiting (-1,2). Compatible with inputs for `plot_directed`. Vector input is supported either directly calling the function with a value for each edge in `E(graph)` or as an edge "attribute" in the igraph object (using `E(g)$state <- states`).

Value

An integer matrix indicating the resolved state (activating or inhibiting for each edge or path between nodes)

Author(s)

Tom Kelly <`tom.kelly@riken.jp`>

See Also

See also `generate_expression` for computing the simulated data, `make_sigma` for computing the Sigma (Σ) matrix, and `make_distance` for computing distance from a graph object.

See also `plot_directed` for plotting graphs or `heatmap.2` for plotting matrices.

See also `make_laplacian`, `make_commonlink`, or `make_adjmatrix` for computing input matrices.

See also `igraph` for handling graph objects.

Other graphsim functions: `generate_expression()`, `make_adjmatrix`, `make_commonlink`, `make_distance`, `make_laplacian`, `make_sigma`, `plot_directed()`

Other generate simulated expression functions: `generate_expression()`, `make_distance`, `make_sigma`

Examples

```
# construct a synthetic graph module
library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)

 # compute state matrix for toy example
state_matrix <- make_state_matrix(graph_test)

# construct a synthetic graph network
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)

# compute state matrix for toy network
graph_structure_state_matrix <- make_state_matrix(graph_structure)
graph_structure_state_matrix

# compute state matrix for toy network with inhibitions
edge_state <- c(1, 1, -1, 1, 1, 1, 1, -1)
# edge states are a variable
graph_structure_state_matrix <- make_state_matrix(graph_structure, state = edge_state)
graph_structure_state_matrix
```

```
# compute state matrix for toy network with inhibitions
E(graph_structure)$state <- c(1, 1, -1, 1, 1, 1, 1, -1)
# edge states are a graph attribute
graph_structure_state_matrix <- make_state_matrix(graph_structure)
graph_structure_state_matrix

library("igraph")
graph_test_edges <- rbind(c("A", "B"), c("B", "C"), c("B", "D"))
graph_test <- graph.edgelist(graph_test_edges, directed = TRUE)
state_matrix <- make_state_matrix(graph_test)

# import graph from package for reactome pathway
# TGF-\eqn{\Beta} receptor signaling activates SMADs (R-HSA-2173789)
TGFBeta_Smad_graph <- identity(TGFBeta_Smad_graph)

# compute sigma (\eqn{\Sigma}) matrix from geometric distance directly from TGF-\eqn{\Beta} pathway
TFGBeta_Smad_state <- E(TGFBeta_Smad_graph)$state
table(TFGBeta_Smad_state)
# states are edge attributes
state_matrix_TFGBeta_Smad <- make_state_matrix(TGFBeta_Smad_graph)
# visualise matrix
library("gplots")
heatmap.2(state_matrix_TFGBeta_Smad , scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))

# compare the states to the sign of expected correlations in the sigma matrix
sigma_matrix_TFGBeta_Smad_inhib <- make_sigma_mat_dist_graph(TGFBeta_Smad_graph,
                                                              cor = 0.8,
                                                              absolute = FALSE)
# visualise matrix
heatmap.2(sigma_matrix_TFGBeta_Smad_inhib,
          scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))

# compare the states to the sign of final correlations in the simulated matrix
TFGBeta_Smad_data <- generate_expression(100, TGFBeta_Smad_graph, cor = 0.8)
heatmap.2(cor(t(TFGBeta_Smad_data)), scale = "none", trace = "none",
          dendrogram = "none", Rowv = FALSE, Colv = FALSE,
          col = colorpanel(50, "blue", "white", "red"))
```

`Pi3K_AKT_graph` *PI3K/AKT activation*

Description

Reactome pathway R-HSA-198203 for the interactions in the phosphoinositide-3-kinase activation of Protein kinase B (PKB), also known as Akt

Usage

```
Pi3K_AKT_graph
```

Format

A graph object of 275 vertices and 21106 edges:

V gene symbol (human)

E directed relationship for pathway

state type of relationship (activating or inhibiting) as edge attribute

Source

PathwayCommons https://reactome.org/content/detail/R-HSA-198203

`Pi3K_graph` *PI3K Cascade*

Description

Reactome pathway R-HSA-109704 for the interactions in the phosphoinositide-3-kinase cascade

Usage

```
Pi3K_graph
```

Format

A graph object of 35 vertices and 251 edges:

V gene symbol (human)

E directed relationship for pathway

state type of relationship (activating or inhibiting) as edge attribute

Source

PathwayCommons https://reactome.org/content/detail/R-HSA-109704

plot_directed *Extensions to igraph for Customising plots*

Description

Functions to plot_directed or graph structures including customised colours, layout, states, arrows. Uses graphs functions as an extension of igraph. Designed for plotting directed graphs.

Usage

```
plot_directed(
  x,
  state = NULL,
  labels = NULL,
  layout = layout.fruchterman.reingold,
  cex.node = 1,
  cex.label = 0.75,
  cex.arrow = 1.25,
  cex.main = 0.8,
  cex.sub = 0.8,
  arrow_clip = 0.075,
  pch = 21,
  border.node = "grey33",
  fill.node = "grey66",
  col.label = NULL,
  col.arrow = NULL,
  main = NULL,
  sub = NULL,
  xlab = "",
  ylab = "",
  frame.plot = F,
  ...
)
```

Arguments

x	An igraph object. Must be directed with known states.
state	character or integer. Defaults to "activating" if no "state" edge attribute found. May be applied a scalar across all edges or as a vector for each edge respectively. Accepts non-integer values for weighted edges provided that the sign indicates whether links are activating (positive) or inhibiting (negative). May also be entered as text for "activating" or "inhibiting" or as integers for activating (0,1) or inhibiting (-1,2). Compatible with inputs for make_state_matrix or generate_expression_graph in the graphsim package https://github.com/TomKellyGenetics/graphsim. Vector input is supported
labels	character vector. For labels to plot nodes. Defaults to vertex names in graph object. Entering "" would yield unlabelled nodes.

`layout`	function. Layout function as selected from `layout_`. Defaults to layout.fruchterman.reingold. Alternatives include layout.kamada.kawai, layout.reingold.tilford, layout.sugiyama, and layout.davidson.harel. A 2-column layout matrix giving x and y co-ordinates of each node can be given.
`cex.node`	numeric. Defaults to 1.
`cex.label`	numeric. Defaults to 0.75.
`cex.arrow`	numeric Defaults to 1.25. May take a scalar applied to all edges or a vector with values for each edge respectively.
`cex.main`	numeric. Defaults to 0.8.
`cex.sub`	numeric. Defaults to 0.8.
`arrow_clip`	numeric Defaults to 0.075 (7.5%).
`pch`	parameter passed to plot. Defaults to 21. Recommends using selecting between 21-25 to preserve colour behaviour. Otherwise entire node will inherit border.node as it's colour, in which case a light colour is recommended to see labels.
`border.node`	character. Specifies the colours of node border passed to plot. Defaults to grey33. Applies to whole node shape if pch has only one colour.
`fill.node`	character. Specfies the colours of node fill passed to plot. Defaults to grey66.
`col.label`	character. Specfies the colours of node labels passed to plot. Defaults to par("fg").
`col.arrow`	character. Specfies the colours of arrows passed to plot. Defaults to par("fg"). May take a scalar applied to all edges or a vector with colours for each edge respectively.
`main`, `sub`, `xlab`, `ylab`	Plotting parameters to specify plot titles or axes labels
`frame.plot`	logical. Whether to frame plot with a box. Defaults to FALSE.
`...`	arguments passed to plot

Value

base R graphics

Author(s)

Tom Kelly <`tom.kelly@riken.jp`>

See Also

See also `generate_expression` for computing the simulated data, `make_sigma` for computing the Sigma (Σ) matrix, `make_distance` for computing distance from a graph object, `make_state` for resolving inhibiting states.

See also `heatmap.2` for plotting matrices.

See also `make_laplacian`, `make_commonlink`, or `make_adjmatrix` for computing input matrices.

See also `igraph` for handling graph objects and `plot.igraph` for base R `plot` methods.

Other graphsim functions: `generate_expression()`, `make_adjmatrix`, `make_commonlink`, `make_distance`, `make_laplacian`, `make_sigma`, `make_state`

Examples

```
# generate example graphs
library("igraph")
graph_structure_edges <- rbind(c("A", "C"), c("B", "C"), c("C", "D"), c("D", "E"),
                               c("D", "F"), c("F", "G"), c("F", "I"), c("H", "I"))
graph_structure <- graph.edgelist(graph_structure_edges, directed = TRUE)

# plots with igraph defaults
plot(graph_structure, layout = layout.fruchterman.reingold)
plot(graph_structure, layout = layout.kamada.kawai)

# plots with scalar states
plot_directed(graph_structure, state="activating")
plot_directed(graph_structure, state="inhibiting")

# plots with vector states
plot_directed(graph_structure, state = c(1, 1, 1, 1, -1, 1, 1, 1))
plot_directed(graph_structure, state = c(1, 1, -1, 1, -1, 1, -1, 1))
plot_directed(graph_structure, state = c(1, 1, -1, 1, 1, 1, 1, -1))

# plots states with graph attributes
E(graph_structure)$state <- 1
plot_directed(graph_structure)
E(graph_structure)$state <- c(1, 1, -1, 1, -1, 1, -1, 1)
plot_directed(graph_structure)

# plot layout customised
plot_directed(graph_structure, state=c(1, 1, -1, 1, -1, 1, -1, 1), layout = layout.kamada.kawai)
```

`RAF_MAP_graph` *#' RAF/MAP kinase cascade*

Description

Reactome pathway R-HSA-5673001 for the interactions in the RAF/MAP kinase cascade

Usage

```
RAF_MAP_graph
```

Format

A graph object of 17 vertices and 121 edges:

V gene symbol (human)

E directed relationship for pathway

Source

PathwayCommons https://reactome.org/content/detail/R-HSA-5673001

`TGFBeta_Smad_graph` *TGF-β receptor signaling activates SMADs*

Description

Reactome pathway R-HSA-2173789 for the interactions in the TGF-β receptor signaling activates SMADs

Usage

```
TGFBeta_Smad_graph
```

Format

A graph object of 32 vertices and 173 edges:

V gene symbol (human)

E directed relationship for pathway

state type of relationship (activating or inhibiting) as edge attribute

Source

PathwayCommons https://reactome.org/content/detail/R-HSA-2173789

Index