Package 'TraRe'

May 23, 2022

Title Transcriptional Rewiring

Version 1.5.0

URL https://github.com/ubioinformat/TraRe/tree/master

Description TraRe (Transcriptional Rewiring) is an R package which contains the necessary tools to carry out several functions.
Identification of module-based gene regulatory networks (GRN); score-based classification of these modules via
a rewiring test; visualization of rewired modules to analyze condition-based GRN deregulation and drop out genes
recovering via cliques methodology. For each tool, an html report can be generated containing useful information about
the generated GRN and statistical data about the performed tests. These tools have been developed considering sequenced data (RNA-Seq).

Depends R (>= 4.1)

Imports hash, ggplot2, stats, methods, igraph, utils, glmnet, vbsr, grDevices, gplots, gtools, pvclust, R.utils, dqrng, SummarizedExperiment, BiocParallel, matrixStats

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

biocViews GeneRegulation, RNASeq, GraphAndNetwork, Bayesian, GeneTarget, Classification

Suggests knitr, rmarkdown, BiocGenerics, RUnit, BiocStyle

VignetteBuilder knitr

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

git_branch master

git_last_commit 6e8284d

git_last_commit_date 2022-04-26

Date/Publication 2022-05-23

Author Jesus De La Fuente Cedeño [aut, cre, cph]
(<https://orcid.org/0000-0003-1856-2469>),
Mikel Hernaez [aut, cph, ths] (<https://orcid.org/0000-0003-0443-2305>),
Charles Blatti [aut, cph] (<https://orcid.org/0000-0002-4683-6271>)

Maintainer Jesus De La Fuente Cedeño <jdelafuentec@unav.es>

R topics documented:

create_html_summary 2
generatecliques 4
html_from_graph 6
LINKER_run 7
LINKER_runPhase1 9
LINKER_runPhase2 12
NET_run 13
plot_igraph 16
preparerewiring 18
rewiring_test 20
runrewiring 21

Index **22**

`create_html_summary` *Create HTML summary*

Description

Given results of Linker runs and other annotations, builds an interconnected html website that summarizes graph edges by support and ChIP evidence (if provided).

Usage

```
create_html_summary(
  rfiles,
  tagstr,
  mapfile,
  outdir = paste0(tempdir(), "/"),
  evidfile
)
```

Arguments

`rfiles,`	character vector of the string file names where results of linker runs have been saved using saveRDS, e.g., linkerResult <- LINKER_run(...), saveRDS(linkerResult, file = 'StringFileName').
`tagstr,`	a string name to tag related html summary results

mapfile, a string name of file that contains identifier and regulator status information, must contain the columns 'uniq_isos' - contains name of rows in LINKER_run `lognorm_est_counts` input matrix 'iso_ensts' - transcript ids, but can be left blank 'iso_ensgs' - contains gene identifiers that match identifiers available in the evidfile 'iso_ensgvs' - versioned gene identifiers, can be left blank 'iso_gnames' - gene name for display 'iso_descs' - extra information about row, can be left blank 'filtered' - whether to include row, can be left blank 'regulator' - whether row is regulator or target (has values 1 or 0 respectively)

outdir, a string name of directory location to save html

evidfile, a string name of file that contain ChIP evidence information, must be a matrix where the column names are the 'iso_ensgs' names of the targets the row names are the 'iso_ensgs' names of the regulators the matrix values are -1: meaning that information is missing, the regulator was not chipped, or the gene names were not mapped 0: meaning that the ChIP'ed regulator did not have a peak in the targets genomic region (gene body +/- 20KB) positive integer: number of peaks of the ChIP'ed regulator in the targets genomic region (gene body +/- 20KB)

Value

allsummaries, a data frame that contains for each module and graph the number of graph edges at each possible level of support and the percentage of cumulative edges with each type of ChIP evidence

Examples

```
## We are going to use files from the example folder.
## `create_html_summary()` only needs the paths, so thats what
## we are giving it.

evidpath <- paste0(system.file('extdata',package='TraRe'),'/ChIP',
                   '/Tumor_OV50_intersectBed.weighted_evidence.txt')

rfiles <- c(paste0(system.file('extdata',package='TraRe'),'/ChIP',
                   '/Tumor_OV50.tar8855_reg638.VBSR.m100_b10.rds'))

tagstr <- 'Tumor_OV50.tar8855_reg638'

mapfile <- paste0(system.file('extdata',package='TraRe'),'/ChIP',
                   '/Tumor_OV50.gene_info.txt')

## We are going to create a folder for this example
## If u want to keep create_html_summary output,
## do not run the last line.

##By default, the output directory will be paste0(tempdir(),'/')
##For this example, we will generate it in the working directory.

create_html_summary(rfiles,tagstr,mapfile,evidfile=evidpath)
unlink(paste0(getwd(),'/',tagstr),recursive = TRUE)
```

generatecliques *Cliques generation and plot displaying.*

Description

By the way the LINKER method works, some highly-correlated driver genes (TFs) may be dropped from the resultant model, as the role they play at the GRN inference process is very similar. Due to this, we propose a method based on cliques (Fully Connected Networks) to recover those dropped drivers.

Usage

```
generatecliques(
  dataset = NULL,
  nassay = 1,
  method = "pearson",
  correlationth = 0.6,
  sparsecorrmatrix = TRUE,
  numcliques = "All",
  mandatorygenes = c(),
  selection = 1
)

preparedata(dataset, method)

generategraph(correlationth, sparsecorrmatrix, pdoutput)

selectmethod(selection, ggoutput, pdoutput)

plotcliques(ml, pdobject, sortparameter_ix_numcliques, smobject, numcliques)
```

Arguments

dataset	input expression file with genes as rows and samples as columns.
nassay	if SummarizedExperiment object is passed as input to dataset, name of the assay containing the desired matrix. Default: 0
method	method to use in the correlation matrix generation (see stats:cor). Default: 'pearson'
correlationth	threshold to consider edge exists. Default: 0.6
sparsecorrmatrix	boolean variable specifying whether to set to 0 values below threshold or not. Default: TRUE
numcliques	number of cliques to be generated. Default: 'All'
mandatorygenes	array of gene names which is mandatory to include in the returned cliques. Default: c() (none)

selection	integer selecting method. The available options are: 1 - Maximize Genes/Clique, 2 - Maximize Median Correlation Value/Clique, 3- Maximize Avg Variance Correlation Value/Clique or 4 - Maximize Sum(option two, option three).
pdoutput	output from preparedata()
ggoutput	output from generategraph()
ml	cliques from smobject
pdobject	output from preparedata()
sortparameter_ix_numcliques	parameter for sorting cliques.
smobject	output from selectmethod()

Details

First, `preparedata()` prepares the correlation matrix and the hash table for future uses. Then, `generategraph()` generates an igraph object from genes correlation (threshold dependant) adjacency matrix. After that, `selectmethod()` chooses method in order to remove duplicities generated from the igraph::max_cliques method. Finally, `generatecliques()` generates all the cliques, containing fully connected networks of genes per clique.

Value

List containing the plot generated and a list with all the generated cliques.

Examples

```
## Suppose we want to recover the drivers LINKER may have dropped out.
## This method allows to group the highly correlated (above `correlationth`)
## driver genes so after the GRN generation, we know that if a particular
## driver gene is taking part of a relationship inside that GRN, all the
## genes inside this group may be also taking part of the same relationship
## due to the high correlation. Note that the large this threshold is, the
## surer we are about this affirmation.


## For this example, we will only work with the driver genes of a example dataset.

dataset <- readRDS(paste0(system.file('extdata',package='TraRe'),
                  '/tfs_linker_example_eg.rds'))

## Lets select the generated dataset, as the rest of parameters are set by default.

clioutput <- generatecliques(dataset = dataset)
```

`html_from_graph` *Html generation*

Description

From input Gene Regulatory Network, an html is generated containing a table with driver to target phenotype dependent relationships. Brief summary containing drivers normalized xor sum, which is the ratio between drivers-targets connections present only in one of both phenotypes over all possible connections, and cliques, which are driver genes that are highly correlated (over a user-decision threshold), and may have been lost during the fitting process of the LINKER method.

Usage

```
html_from_graph(
  gpath = NULL,
  wpath = paste0(tempdir()),
  user_mode = TRUE,
  cliquesbool = TRUE,
  ...
)
```

Arguments

`gpath`	path to the graph object ('refinedsumm.rds'). (RDS format)
`wpath`	writing path, where the html and txts file will be saved. (Default: temp directory)
`user_mode`	boolean indicating if this function is called from user or internaly. (Default: TRUE)
`cliquesbool`	indicating if cliques method should be added to the summary table. (Default: TRUE)
`...`	every argument you should pass to generatecliques() in case cliquesbool is TRUE.

Value

Html and txts containing the mentioned files.

Examples

```
## For this example, we are going to use a generated 'refinedsumm.rds' from the su2c dataset
## (see vignette of TraRe for more information), which is at the external data folder
## of this package.

gpath <- paste0(system.file('extdata',package='TraRe'),'/refinedsumm.rds')
wpath <- paste0(system.file('extdata',package='TraRe'))

## We are going to use the drivers dataset from the external data folder as well.
## For more information about generatecliques() please check the help page of it.
```

```
dataset<-readRDS(paste0(system.file('extdata',package='TraRe')
                 ,'/tfs_linker_example_eg.rds'))
html_from_graph(gpath=gpath,wpath=wpath,dataset=dataset)
```

LINKER_run *GRN inference via selected model.*

Description

Gene Regulatory Network inference via model selection. Consists of two phases, `LINKER_runPhase1()` and `LINKER_runPhase2()`. Help them for more information.

Usage

```
LINKER_run(
  lognorm_est_counts,
  target_filtered_idx,
  regulator_filtered_idx,
  nassay = 1,
  regulator = "regulator",
  link_mode = c("VBSR", "LASSOmin", "LASSO1se", "LM"),
  graph_mode = c("VBSR", "LASSOmin", "LASSO1se", "LM"),
  module_rep = "MEAN",
  NrModules = 100,
  corrClustNrIter = 100,
  Nr_bootstraps = 10,
  FDR = 0.05,
  NrCores = 1
)
```

Arguments

`lognorm_est_counts`
: Matrix of log-normalized estimated counts of the gene expression data (Nr Genes x Nr samples) or SummarizedExperiment object.

`target_filtered_idx`
: Index array of the target genes on the lognorm_est_counts matrix if SummarizedExperiment object is not provided.

`regulator_filtered_idx`
: Index array of the regulatory genes on the lognorm_est_counts matrix if SummarizedExperiment object is not provided.

`nassay`
: if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the assay containing the desired matrix. Default: 1 (first item in assay's list).

regulator	if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the rowData() variable to build target_filtered_idx and regulator_filtered_idx. This variable must be one for driver genes and zero for target genes. Default: 'regulator'
link_mode	Chosen method(s) to link module eigengenes to regulators. The available options are 'VBSR', 'LASSOmin', 'LASSO1se' and 'LM'. By default, all methods are chosen.
graph_mode	Chosen method(s) to generate the edges in the bipartite graph. The available options are 'VBSR', 'LASSOmin', 'LASSO1se' and 'LM'. By default, all methods are chosen.
module_rep	Method selected for use. Default set to MEAN.
NrModules	Number of modules that are a priori to be found (note that the final number of modules discovered may differ from this value). By default, 100 modules.
corrClustNrIter	output from preparedata(). By default, 100.
Nr_bootstraps	Number of bootstrap of Phase I. By default, 10.
FDR	The False Discovery Rate correction used for the enrichment analysis. By default, 0.05.
NrCores	Nr of computer cores for the parallel parts of the method. Note that the parallelization is NOT initialized in any of the functions. By default, 2.

Value

List containing the GRN raw results, GRN modules and GRN graphs.

Examples

```
 ## For this example, we are going to join 60 drivers and
 ## 200 targets genes from the example folder.

drivers <- readRDS(paste0(system.file('extdata',package='TraRe'),'/tfs_linker_example.rds'))
targets <- readRDS(paste0(system.file('extdata',package='TraRe'),'/targets_linker_example.rds'))

 lognorm_est_counts <- as.matrix(rbind(drivers,targets))

## We create the index for drivers and targets in the log-normalized gene expression matrix.

 R<-60
 T<-200

 regulator_filtered_idx <- seq_len(R)
 target_filtered_idx <- R+c(seq_len(T))


 ## We recommend to use the default values of the function.
 ## For the sake of time, we will select faster (and worse) ones.

linkeroutput <- LINKER_run(lognorm_est_counts,target_filtered_idx=target_filtered_idx,
```

```
        regulator_filtered_idx=regulator_filtered_idx,
link_mode='LASSOmin',graph_mode='LM',NrModules=5,Nr_bootstraps=1,
        NrCores=2,corrClustNrIter=10)
```

LINKER_runPhase1 *Phase I : module generation*

Description

Run first phase of the linker method where K modules of similarly expressed target genes and relate them to a linear combination of very few regulators, according to the selected model. `LINKER_init()` evaluate kmeans on a train set to generate a initial set of clusters containing drivers and target genes. `LINKER_ReassignGenesToClusters()` reassigning genes based on closed match to new regulatory programs. This functions takes place inside the linkerrun function, so it is not recommended to run it on its own. `LINKER_corrClust()` go through two steps within a loop, learning regulatory program of modules and reassigning genes. `LINKER_extract_modules()` extract all the modules, genes and relevant information. `LINKER_EvaluateTestSet()` fits the selected model with the test data. `LINKER_LearnRegulatoryPrograms()` learns the regulatory program of the modules.

Usage

```
LINKER_runPhase1(
  lognorm_est_counts,
  target_filtered_idx,
  regulator_filtered_idx,
  nassay = 1,
  regulator = "regulator",
  NrModules,
  Lambda = 1e-04,
  alpha = 1 - 1e-06,
  pmax = 10,
  mode = "VBSR",
  used_method = "MEAN",
  NrCores = 1,
  corrClustNrIter = 100,
  Nr_bootstraps = 1
)

LINKER_init(
  MA_matrix_Var,
  RegulatorData,
  NrModules,
  NrCores = 3,
  corrClustNrIter = 21,
```

```
  Parameters
)

LINKER_ReassignGenesToClusters(
  Data,
  RegulatorData,
  Beta,
  Clusters,
  NrCores = 1
)

LINKER_corrClust(LINKERinit)

LINKER_extract_modules(results)

LINKER_EvaluateTestSet(
  LINKERresults,
  MA_Data_TestSet,
  RegulatorData_TestSet,
  used_method = "MEAN"
)

LINKER_LearnRegulatoryPrograms(
  Data,
  Clusters,
  RegulatorData,
  Lambda,
  alpha,
  pmax,
  mode,
  used_method = "MEAN",
  NrCores = 1
)
```

Arguments

lognorm_est_counts	Matrix of log-normalized estimated counts of the gene expression data (Nr Genes x Nr samples)
target_filtered_idx	Index array of the target genes on the lognorm_est_counts matrix if SummarizedExperiment object is not provided.
regulator_filtered_idx	Index array of the regulatory genes on the lognorm_est_counts matrix if SummarizedExperiment object is not provided.
nassay	if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the assay containing the desired matrix. Default: 1 (first item in assay's list).

regulator	if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the rowData() variable to build target_filtered_idx and regulator_filtered_idx. This variable must be one for driver genes and zero for target genes. Default: 'regulator'
NrModules	Number of modules that are a priori to be found (note that the final number of modules discovered may differ from this value). By default, 100 modules.
Lambda	Lambda variable for Lasso models.
alpha	Alpha variable for Lasso models.
pmax	Maximum numbers of regulators that we want.
mode	Chosen method(s) to link module eigengenes to regulators. The available options are 'VBSR', 'LASSOmin', 'LASSO1se' and 'LM'. Default set to 'VBSR'
used_method	Method selected for use. Default set to MEAN.
NrCores	Nr of computer cores for the parallel parts of the method. Note that the parallelization is NOT initialized in any of the functions. By default, 2.
corrClustNrIter	Number of iteration for the phase I part of the method.
Nr_bootstraps	Number of bootstrap of Phase I. By default, 1.
MA_matrix_Var	Matrix of log-normalized estimated counts of the gene expression data, centered and scaled, containing only the train samples.
RegulatorData	Expression matrix containing only the regulators of the train samples.
Parameters	List of parameters containig lambda, pmax, alpha, mode and used method.
Data	Matrix of log-normalized estimated counts of the gene expression data, centered and scaled, containing only the train samples.
Beta	Coefficient on which the decision of reassigning genes is based.
Clusters	Clusters generated from the linkerinit function.
LINKERinit	Initialization object obtained from `LINKER_init()`.
results	Matrix of log-normalized estimated counts of the gene expression data (Nr Genes x Nr samples).
LINKERresults	List containing the number of clusters, regulatoryprogram, name of regulators and all genes and module membership.
MA_Data_TestSet	Matrix of log-normalized estimated counts of the gene expression data, centered and scaled, containing only the test samples.
RegulatorData_TestSet	Expression matrix containing only the regulators of the test samples.

Value

igraph object containing the modules containing the related drivers and targets within bootstraps.

Examples

```
 ## This example is very similar to the `LINKER_run()` function.
 ## Again, we are going to join drivers and targets genes to create the working dataset.

drivers <- readRDS(paste0(system.file('extdata',package='TraRe'),'/tfs_linker_example.rds'))
targets <- readRDS(paste0(system.file('extdata',package='TraRe'),'/targets_linker_example.rds'))

 lognorm_est_counts <- rbind(drivers,targets)
## We create the index for drivers and targets in the log-normalized gene expression matrix.

 R<-60
 T<-200

 regulator_filtered_idx <- seq_len(R)
 target_filtered_idx <- R+c(seq_len(T))

 ## We recommend to use the default values of the function.
 ## For the sake of time, we will select faster (and worse) ones.

linkeroutput <- LINKER_runPhase1(lognorm_est_counts,target_filtered_idx=target_filtered_idx,
                           regulator_filtered_idx=regulator_filtered_idx, NrModules=2,
                         mode='LASSOmin',NrCores=2, corrClustNrIter=10,Nr_bootstraps=1)
```

`LINKER_runPhase2` *Phase II : bipartitive graphs generation*

Description

Run second phase of the linker method where a bipartitive graph is generated from the phase I output. A bipartite graph is a set of graph nodes decomposed into two disjoint sets such that no two graph nodes within the same set are adjacent.

Usage

```
LINKER_runPhase2(modules, Data, NrCores, mode = "VBSR", alpha = 1 - 1e-06)
```

Arguments

`modules`	Modules obtained from the phase I linker output.
`Data`	Matrix of log-normalized estimated counts of the gene expression data (Nr Genes x Nr samples)
`NrCores`	Nr of computer cores for the parallel parts of the method. Note that the parallelization is NOT initialized in any of the functions. By default, 2.
`mode`	Chosen method(s) to link module eigengenes to regulators. The available options are 'VBSR', 'LASSOmin', 'LASSO1se' and 'LM'. By default, all methods are chosen.
`alpha`	alpha parameter if a LASSO model is chosen.

Value

igraph object containing the related drivers and targets in the form of a bipartitive graph.

Examples

```
## We are going to proceed in the same manner as in the `liner_runphaseone()` example
## but we start at the end of it, loading the output from the example folder.

phaseone <- readRDS(paste0(system.file('extdata',package='TraRe'),
                           '/linker_phaseone_example.rds'))

## We are loading drivers and targets to generate lognorm_est_counts, as we need it
## for the phase 2.

drivers <- readRDS(paste0(system.file('extdata',package='TraRe'),
                                     '/tfs_linker_example.rds'))
targets <- readRDS(paste0(system.file('extdata',package='TraRe'),
                                     '/targets_linker_example.rds'))

lognorm_est_counts <- as.matrix(rbind(drivers,targets))

## Now we proceed to call `LINKER_runPhase2()`.
## We first, we need to extract modules from the `LINKER_runPhase1()` output.

modules_phaseone<-LINKER_extract_modules(phaseone)

## Now we generate the bipartitive graph from the extracted modules


graph <- LINKER_runPhase2(modules=modules_phaseone,Data=lognorm_est_counts,
                          NrCores=1,mode='LM')
```

NET_run *Single gene network approaches.*

Description

NET_run() generate a single GRN. NET_compute_graph_all_LASSO1se() defines the statistics of drivers and targets to be the Lasso method, choosing 1 standard error from the minimum RSS. NET_compute_graph_all_LASSOmin() uses Lasso method, choosing the minimum RSS point. NET_compute_graph_all_LM() uses a linear model and NET_compute_graph_all_VBSR() uses a Variational Bayes Spike Regression.

Usage

```
NET_run(
  lognorm_est_counts,
```

```
  target_filtered_idx,
  regulator_filtered_idx,
  nassay = 1,
  regulator = "regulator",
  graph_mode = c("VBSR", "LASSOmin", "LASSO1se", "LM"),
  FDR = 0.05,
  NrCores = 1
)

NET_compute_graph_all_LASSO1se(
  lognorm_est_counts,
  regulator_filtered_idx,
  target_filtered_idx,
  alpha = 1 - 1e-06,
  NrCores = 1
)

NET_compute_graph_all_LASSOmin(
  lognorm_est_counts,
  regulator_filtered_idx,
  target_filtered_idx,
  alpha = 1 - 1e-06,
  NrCores = 1
)

NET_compute_graph_all_LM(
  lognorm_est_counts,
  regulator_filtered_idx,
  target_filtered_idx,
  NrCores = 1
)

NET_compute_graph_all_VBSR(
  lognorm_est_counts,
  regulator_filtered_idx,
  target_filtered_idx,
  NrCores = 1
)
```

Arguments

`lognorm_est_counts`
: Matrix of log-normalized estimated counts of the gene expression data (Nr Genes x Nr samples)

`target_filtered_idx`
: Index array of the target genes on the lognorm_est_counts matrix if SummarizedExperiment object is not provided.

`regulator_filtered_idx`
: Index array of the regulatory genes on the lognorm_est_counts matrix if Sum-

	marizedExperiment object is not provided.
nassay	if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the assay containing the desired matrix. Default: 1 (first item in assay's list).
regulator	if SummarizedExperiment object is passed as input to lognorm_est_counts, name of the rowData() variable to build target_filtered_idx and regulator_filtered_idx. This variable must be one for driver genes and zero for target genes. Default: 'regulator'
graph_mode	Chosen method(s) to generate the edges in the bipartite graph. The available options are 'VBSR', 'LASSOmin', 'LASSO1se' and 'LM'. By default, all methods are chosen.
FDR	The False Discovery Rate correction used for the enrichment analysis. By default, 0.05.
NrCores	Nr of computer cores for the parallel parts of the method. Note that the parallelization is NOT initialized in any of the functions. By default, 3.
alpha	feature selection parameter in case of a LASSO model to be chosen.

Value

List containing the GRN graphs.

Examples

```
 ## Assume we have run the rewiring method and we have discovered a rewired module.
 ## After we have selected the drivers and targets from that modules, we can build
 ## a single GRN to study it separately.


 ## For this example, we are going to join 60 drivers and
 ## 200 targets genes from the example folder.

drivers <- readRDS(paste0(system.file('extdata',package='TraRe'),'/tfs_linker_example.rds'))
targets <- readRDS(paste0(system.file('extdata',package='TraRe'),'/targets_linker_example.rds'))

 lognorm_est_counts <- as.matrix(rbind(drivers,targets))

## We create the index for drivers and targets in the log-normalized gene expression matrix.

 R<-60
 T<-200

 regulator_filtered_idx <- seq_len(R)
 target_filtered_idx <- R+c(seq_len(T))

 ## We recommend VBSR (rest of parameters are set by default)
 graph <- NET_run(lognorm_est_counts,target_filtered_idx=target_filtered_idx,
                  regulator_filtered_idx=regulator_filtered_idx,graph_mode='VBSR')
```

plot_igraph *Plotting functions for Gene Regulatory Network.*

Description

Collection of functions for generating graphs layouts to plot GRN obtained from NET_run() method. return_layout() generates a layout from the graph object returned by NET_run() and return_layout_phenotype() plots targets according to the t-statistic from the differential expression analysis of the desired phenotype. plot_igraph() takes in the igraph object and generated layout and generates plot.

Usage

```
plot_igraph(mygraph = NULL, mytitle = "", titlecol = "black", mylayout = NULL)

return_layout(regs = NULL, targets = NULL, namehash = NULL)

return_layout_phenotype(
  regs = NULL,
  targets = NULL,
  varfile = NULL,
  namehash = NULL
)

orderGraphWeights(graph, edgelist)
```

Arguments

mygraph	igraph object returned from NET_run().
mytitle	Desired tittle.
titlecol	Color for the tittle.
mylayout	desired layout.
regs	regulators name list
targets	targets name list
namehash	list containing the drivers genes as names and transcripts as values. If only genes are required, leave it empty.
varfile	two column file containing, gene names as rows, t-statistic from the differential expression analysis of the desired phenotype column and a boolean variable for regulator (1) - no regulator (0) column.
graph	igraph object
edgelist	list containing the edges of the igraph object.

Value

plot of the desired single GRN using a specific layout.

Examples

```
 ## Assume we have run the rewiring method and the `NET_run()` method to generate the
 ## igraph object. We are going to generate and plot both layouts for the example.
 ## We are going to generate all the files we need except for the igraph object, which
 ## is included as an example file in this package.

 ## We load the igraph object that we generated from the `NET_run()` example.
 ## Note: the igraph object is inside the list `NET_run()` generates.

 graph <- readRDS(paste0(system.file('extdata',package='TraRe'),
                 '/graph_netrun_example.rds'))$graphs$VBSR

 ## We first generate the normal layout for the plot.
 ## We need the drivers and target names.

drivers <- readRDS(paste0(system.file('extdata',package='TraRe'),'/tfs_linker_example.rds'))
 drivers_n <- rownames(drivers)

targets <- readRDS(paste0(system.file('extdata',package='TraRe'),'/targets_linker_example.rds'))
 targets_n <- rownames(targets)

## As for this example we are working at gene level (we dont have transcripts inside genes),
 ## we will generate a dictionary with genes as keys and values (see param `namehash`)

 normal_layout <- return_layout(drivers_n,targets_n)

 ## We now generate the phenotype layout and the `varfile` we ned for this layout.
 ## (I leave here a way to generate) We need to separate our expression matrix by
 ## a binary phenotype, for this case, i will consider the first 40 samples are
 ## responding to a treatment (R) and the rest not (NR).

 gnames <- c(drivers_n,targets_n)
 expmat <-rbind(drivers,targets)

 phenotype <- utils::read.delim(paste0(system.file('extdata',package='TraRe'),
                                '/phenotype_rewiring_example.txt'))

 expmat_R <- expmat[,phenotype$Class=='R']
 expmat_NR <- expmat[,phenotype$Class=='NR']


 varfile <- t(as.matrix(sapply(gnames,
            function(x) c(stats::t.test(expmat_R[x,],expmat_NR[x,])$statistic,
            if(x%in%drivers_n) 1 else 0))))

 colnames(varfile)<-c('t-stat','is-regulator')

 phenotype_layout <- return_layout_phenotype(drivers_n,targets_n,varfile)

 plot_igraph(graph,mytitle='Normal Layout',titlecol='black',mylayout=normal_layout)
plot_igraph(graph,mytitle='Phenotype Layout',titlecol='black',mylayout=phenotype_layout)
```

preparerewiring *Prepare rewiring data for running the method.*

Description

Prepare neccessary files for running `runrewiring()`

Usage

```
preparerewiring(
  name = "defaultname",
  linker_output_p,
  lognorm_est_counts_p = NULL,
  SEObject_p = NULL,
  gene_info_p = NULL,
  phenotype_p = NULL,
  nassays = 1,
  final_signif_thresh = 0.001,
  regulator_info_col_name = "regulator",
  phenotype_col_name = "Class",
  phenotype_class_vals_string = "NR,R",
  phenotype_class_vals_string_label = "0,1",
  orig_test_perms = 100,
  retest_thresh = 0.08,
  retest_perms = 1000,
  outdir = tempdir(),
  nrcores = 3
)
```

Arguments

`name`	Desired name of the folder which is generated. The chosen threshold will be `paste()` to the folder's name.
`linker_output_p`	Output file from linker function path. RDS format is required.
`lognorm_est_counts_p`	Lognorm counts of the gene expression matrix path.
`SEObject_p`	SummarizedExperiment objects path.
`gene_info_p`	path of a two-column file containing genes and 'regulator' boolean variable.
`phenotype_p`	path of a two-column file containing used samples and Responder or No Responder 'Class' (NR,R).
`nassays`	name of assays in case SummarizedObject is provided.
`final_signif_thresh`	Significance threshold for the rewiring method. The lower the threshold, the restrictive the method.

regulator_info_col_name
: Column name of the gene_info_p. By default, 'regulator'.

phenotype_col_name
: Column name of the phenotype_p. By default, 'Class'.

phenotype_class_vals_string
: Boolean terms of the phenotype_p at the 'Class' column. By default, (NR,R).

phenotype_class_vals_string_label
: Boolean terms of the phenotype_p values at the 'Class' column. By default (0,1)

orig_test_perms
: Initial permutations for first test (default: 100) .

retest_thresh
: Threshold if a second test is performed (default: 0.08) .

retest_perms
: Permutations if a second test is performed (default: 1000) .

outdir
: Directory for the output folder to be located (default: tempdir())

nrcores
: Number of cores to run the parallelization within the rewiring test (default: 3).

Value

Return a list containing: LINKER's output, expression matrix, boolean array from phenotype file, array containing number of c(R,NR) samples, significance threshold and output directory.

Examples

```
## We are going to prepare 4 files that we have in the example folder: the output from LINKER, the
## gene expression matrix, the phenotype file and the gene info file. Note that the LINKER
## output is generated from the gene expression matrix. Note: if rewiring across more than 1 dataset
## is desired, paths will be given as arrays. (i.e. linker_output <- c(path1,path2))


linker_output_p <- paste0(system.file('extdata',package='TraRe'),'/linker_rewiring_example.rds')

lognorm_est_counts_p <- paste0(system.file('extdata',package='TraRe'),
                        '/expression_rewiring_example.txt')

gene_info_p <- paste0(system.file('extdata',package='TraRe'),'/geneinfo_rewiring_example.txt')

phenotype_p <- paste0(system.file('extdata',package='TraRe'),'/phenotype_rewiring_example.txt')

outdir <- system.file('extdata',package='TraRe')

prepared <- preparerewiring(name='example',linker_output_p=linker_output_p,
                      lognorm_est_counts_p=lognorm_est_counts_p,gene_info_p=gene_info_p,
                            phenotype_p=phenotype_p,final_signif_thresh=0.05,
                            nrcores=1,outdir=outdir)
```

`rewiring_test` *Generate stats for the rewiring method.*

Description

When performing the rewiring test, some stats must be generated in order to proceed with the rewiring method. `dave_test()` performs a permutation test from a data matrix and a group membership. This function is used in the `runrewiring()` method.

Usage

```
rewiring_test(x, grp, perm = 500)

rewiring_test_pair_detail(x, grp, perm = 500)
```

Arguments

`x`	data matrix containing subjects as rows and genes as columns.
`grp`	array indicating the subject group membership.
`perm`	number of permutations for the test.

Value

list containing the pvalue associated to the rewiring test, the initial t-stat and the t-stat after the permutation test.

Examples

```
## We are going to generate a random matrix `x` and `grp` for the example.
## We will use 40 samples and 100 genes. First 25 samples will belong to one group
## and the rest (15) to the other.

mat <- matrix(stats::rnorm(40*100),40,100)
group_m <- c(rep(1,25),rep(0,15))

## Note: the `rewiring_test()` works with group membership (1,2) instead of (0,1)

results <- rewiring_test(x=mat,grp=1+group_m)
```

runrewiring *GRN Modules Rewiring Method.*

Description

Gene Regulatory Network modules Rewiring method. It performs a permutation test, (what we call rewiring test) and generates an html report containing a correlation matrix with the higher scores obtained from the rewiring test. This matrix is shown in the way of a heatmap, and its sorted by a hierarchical clustering for better interpretation.

Usage

```
runrewiring(ObjectList)
```

Arguments

ObjectList	Output from preparerewiring()containing some required parameters.

Value

It creates a folder (in tempdir() by default) containing the files explained above.

Examples

```
## Lets assume that we have already generated the ObjectList, we will load it from
## the folder containing the examples files. After this, it is all straight forward.

objectlist <- readRDS(file=paste0(system.file('extdata',package='TraRe'),
                      '/prepared_rewiring_example.rds'))


## We are going to create the folder containing
## the graphs, reports, etc, and then we are deleting it.
## If you want to keep it, do not run the last line.

## We are modifying output directory for this example.
objectlist$outdir <- paste(getwd(),'examplefolder',sep='/')

runrewiring(ObjectList = objectlist)
unlink(objectlist$outdir,recursive = TRUE)
```

Index

create_html_summary, 2

generatecliques, 4
generategraph *(generatecliques)*, 4

html_from_graph, 6

LINKER_corrClust *(LINKER_runPhase1)*, 9
LINKER_EvaluateTestSet *(LINKER_runPhase1)*, 9
LINKER_extract_modules *(LINKER_runPhase1)*, 9
LINKER_init *(LINKER_runPhase1)*, 9
LINKER_LearnRegulatoryPrograms *(LINKER_runPhase1)*, 9
LINKER_ReassignGenesToClusters *(LINKER_runPhase1)*, 9
LINKER_run, 7
LINKER_runPhase1, 9
LINKER_runPhase2, 12

NET_compute_graph_all_LASSO1se *(NET_run)*, 13
NET_compute_graph_all_LASSOmin *(NET_run)*, 13
NET_compute_graph_all_LM *(NET_run)*, 13
NET_compute_graph_all_VBSR *(NET_run)*, 13
NET_run, 13

orderGraphWeights *(plot_igraph)*, 16

plot_igraph, 16
plotcliques *(generatecliques)*, 4
preparedata *(generatecliques)*, 4
preparerewiring, 18

return_layout *(plot_igraph)*, 16
return_layout_phenotype *(plot_igraph)*, 16
rewiring_test, 20
rewiring_test_pair_detail *(rewiring_test)*, 20
runrewiring, 21

selectmethod *(generatecliques)*, 4