Package 'XGR'

March 2, 2016

Type Package

Title Using Ontologies to Increase Interpretability of Genes and SNPs (eg Identified from GWAS and eQTL Mappings)

Version 0.99.0

Date 2016-3-2

Author Julian C Knight group

Maintainer Hai Fang <hfang@well.ox.ac.uk>

Depends R (>= 3.1.0), igraph, dnet

Imports Matrix, RCircos, GenomicRanges

Suggests foreach, doMC

Description Genome-wide association studies (GWAS) identify a wealth body of trait-associated genetic variants (largely SNPs), and expression quantitative trait mappings (eQTLs) help link genetic variants to target genes. We decipher target genes and genetic variants based on known systematic annotations by ontologies. This package supports a wide range of ontologies (covering knowledge on functions, pathways, diseases and phenotypes- in both human and mouse). With a list of genes (or SNPs) to be analysed, users are able to identify the underlying knowledge enriched within them, and to calculate the semantic similarity between their objects.

URL https://github.com/hfang-bristol/XGR

Collate 'xRDataLoader.r'
'xRdWrap.r'
'xFunArgs.r'
'xRd2HTML.r'
'xDAGanno.r'
'xDAGsim.r'
'xConverter.r'
'xEnricher.r'
'xEnricherGenes.r'
'xEnricherSNPs.r'
'xEnricherYours.r'
'xEnrichViewer.r'
'xSocialiser.r'
'xSocialiserGenes.r'
'xSocialiserSNPs.r'
'xCircos.r'
'xSubneter.r'
'xVisNet.r'

License GPL-2

biocViews Bioinformatics

R topics documented:

`xCircos` *Function to visualise semantic similarity results as a circos plot*

Description

`xCircos` is used to visualise the results of similarity analysis as a circos plot.

Usage

```
xCircos(g, entity = c("SNP", "Gene"), top_num = 50, ideogram = T,
chr.exclude = NULL, entity.label.cex = 0.8, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

`g`	an object of class "igraph". It stores semantic similarity results with nodes for genes/SNPs and edges for pair-wise semantic similarity between them
`entity`	the entity of similarity analysis for which results are being plotted. It can be either "SNP" or "Gene"
`top_num`	the top number of simiarity edges to be plotted
`ideogram`	logical to indicate whether chromosome banding is plotted
`chr.exclude`	a character vector of chromosomes to exclude from the plot, e.g. c("chrX", "chrY"). Default is NULL

`entity.label.cex` the font size of genes/SNPs labels. Default is 0.8

`verbose` logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

`RData.location` the characters to tell the location of built-in RData files. See `xRDataLoader` for details

Value

a circos plot with the semantic similarity between input snps/genes represented by the colour of the links

Note

none

See Also

`xSocialiserGenes`, `xSocialiserSNPs`

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)
library(RCircos)
library(GenomicRanges)

# provide genes and SNPs reported in AS GWAS studies
ImmunoBase <- xRDataLoader(RData.customised='ImmunoBase')

# SNP-based similarity analysis using GWAS Catalog traits (mapped to EF)
## Get lead SNPs reported in AS GWAS
example.snps <- names(ImmunoBase$AS$variants)
SNP.g <- xSocialiserSNPs(example.snps, include.LD=NA)
# Circos plot of the EF-based SNP similarity network
#out.file <- "SNP_Circos.pdf"
#pdf(file=out.file, height=12, width=12, compress=TRUE)
xCircos(g=SNP.g, entity="SNP")
#dev.off()

# Gene-based similarity analysis using Disease Ontology (DO)
## Get genes within 10kb away from AS GWAS lead SNPs
example.genes <- names(which(ImmunoBase$AS$genes_variants<=10000))
gene.g <- xSocialiserGenes(example.genes, ontology=c("DO")
# Circos plot of the DO-based gene similarity network
#out.file <- "Gene_Circos.pdf"
#pdf(file=out.file, height=12, width=12, compress=TRUE)
xCircos(g=gene.g, entity="Gene", chr.exclude="chrY")
#dev.off()

## End(Not run)
```

xConverter *Function to convert an object between graph classes*

Description

xConverter is supposed to convert an object between classes 'dgCMatrix' and 'igraph'.

Usage

```
xConverter(obj, from = c("dgCMatrix", "igraph"), to = c("igraph",
"dgCMatrix"), verbose = TRUE)
```

Arguments

obj	an object of class "dgCMatrix" or "igraph"
from	a character specifying the class converted from. It can be one of "dgCMatrix" and "igraph"
to	a character specifying the class converted to. It can be one of "dgCMatrix" and "igraph"
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

an object of class "dgCMatrix" or "igraph"

Note

Conversion is also supported between classes 'dgCMatrix' and 'igraph'

See Also

xRDataLoader

Examples

```
## Not run:
# Conversion between 'dgCMatrix' and 'igraph'
# ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')
g

# convert the object from 'igraph' to 'dgCMatrix' class
s <- xConverter(g, from='igraph', to='dgCMatrix')
s[1:10,1:10]

# convert the object from 'dgCMatrix' to 'igraph' class
ig <- xConverter(s, from="dgCMatrix", to="igraph")
ig

## End(Not run)
```

xDAGanno *Function to generate a subgraph of a direct acyclic graph (DAG) induced by the input annotation data*

Description

xDAGanno is supposed to produce a subgraph induced by the input annotation data, given a direct acyclic graph (DAG; an ontology). The input is a graph of "igraph", a list of the vertices containing annotation data, and the mode defining the paths to the root of DAG. The induced subgraph contains vertices (with annotation data) and their ancestors along with the defined paths to the root of DAG. The annotations at these vertices (including their ancestors) can also be updated according to the true-path rule: those annotated to a term should also be annotated by its all ancestor terms.

Usage

```
xDAGanno(g, annotation, path.mode = c("all_paths", "shortest_paths",
"all_shortest_paths"), true.path.rule = TRUE, verbose = TRUE)
```

Arguments

g	an object of class "igraph" to represent DAG
annotation	the vertices/nodes for which annotation data are provided. It can be a sparse Matrix of class "dgCMatrix" (with variants/genes as rows and terms as columns), or a list of nodes/terms each containing annotation data, or an object of class 'GS' (basically a list for each node/term with annotation data)
path.mode	the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)
true.path.rule	logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

- subg: an induced subgraph, an object of class "igraph". In addition to the original attributes to nodes and edges, the return subgraph is also appended by two node attributes: 1) "anno" containing a list of variants/genes either as original annotations (and inherited annotations; 2) "IC" standing for information content defined as negative 10-based log-transformed frequency of variants/genes annotated to that term.

Note

For the mode "shortest_paths", the induced subgraph is the most concise, and thus informative for visualisation when there are many nodes in query, while the mode "all_paths" results in the complete subgraph.

See Also

xRDataLoader

Examples

```
## Not run:
# 1) SNP-based ontology
# 1a) ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')

# 1b) load GWAS SNPs annotated by EF (an object of class "dgCMatrix" storing a spare matrix)
anno <- xRDataLoader(RData='GWAS2EF')

# 1c) prepare for annotation data
# randomly select 5 terms/vertices (and their annotation data)
annotation <- anno[, sample(1:dim(anno)[2],5)]

# 1d) obtain the induced subgraph according to the input annotation data
# based on shortest paths (i.e. the most concise subgraph induced)
dag <- xDAGanno(g, annotation, path.mode="shortest_paths",
verbose=TRUE)

# 1e) color-code nodes/terms according to the number of annotations
data <- sapply(V(dag)$anno, length)
names(data) <- V(dag)$name
dnet::visDAG(g=dag, data=data, node.info="both")

####################
# Below is for those SNPs annotated by the term called 'ankylosing spondylitis'
# The steps 1a) and 1b) are the same as above
# 1c') prepare for annotation data
# select a term 'ankylosing spondylitis'
terms <- V(g)$term_id[grep('ankylosing spondylitis',V(g)$term_name,
perl=TRUE)]
ind <- which(colnames(anno) %in% terms)
annotation <- lapply(ind, function(x){names(which(anno[,x]!=0))})
names(annotation) <- colnames(anno)[ind]

# 1d') obtain the induced subgraph according to the input annotation data
# based on all possible paths (i.e. the complete subgraph induced)
dag <- xDAGanno(g, annotation, path.mode="all_paths", verbose=TRUE)

# 1e') color-code nodes/terms according to the number of annotations
data <- sapply(V(dag)$anno, length)
names(data) <- V(dag)$name
dnet::visDAG(g=dag, data=data, node.info="both")

############################################################
# 2) Gene-based ontology
# 2a) ig.MP (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.MP')

# 2b) load human genes annotated by MP (an object of class "GS" containing the 'gs' component)
GS <- xRDataLoader(RData='org.Hs.egMP')
anno <- GS$gs # notes: This is a list

# 2c) prepare for annotation data
# randomly select 5 terms/vertices (and their annotation data)
annotation <- anno[sample(1:length(anno),5)]
```

```
# 2d) obtain the induced subgraph according to the input annotation data
# based on shortest paths (i.e. the most concise subgraph induced) but without applying true-path rule
dag <- xDAGanno(g, annotation, path.mode="shortest_paths",
true.path.rule=TRUE, verbose=TRUE)

# 2e) color-code nodes/terms according to the number of annotations
data <- sapply(V(dag)$anno, length)
names(data) <- V(dag)$name
dnet::visDAG(g=dag, data=data, node.info="both")

## End(Not run)
```

`xDAGsim` *Function to calculate pair-wise semantic similarity between input terms based on a direct acyclic graph (DAG) with annotated data*

Description

`xDAGsim` is supposed to calculate pair-wise semantic similarity between input terms based on a direct acyclic graph (DAG) with annotated data. It returns an object of class "igraph", a network representation of input terms. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
xDAGsim(g, terms = NULL, method.term = c("Resnik", "Lin", "Schlicker",
"Jiang", "Pesquita"), fast = T, parallel = TRUE, multicores = NULL,
verbose = T)
```

Arguments

`g`	an object of class "igraph". It must contain a vertex attribute called 'anno' for storing annotation data (see example for howto)
`terms`	the terms/nodes between which pair-wise semantic similarity is calculated. If NULL, all terms in the input DAG will be used for calcluation, which is very prohibitively expensive!
`method.term`	the method used to measure semantic similarity between input terms. It can be "Resnik" for information content (IC) of most informative common ancestor (MICA) (see http://arxiv.org/pdf/cmp-lg/9511007.pdf), "Lin" for 2*IC at MICA divided by the sum of IC at pairs of terms (see http://webdocs.cs.ualberta.ca/~lindek/papers/sim.pdf), "Schlicker" for weighted version of 'Lin' by the 1-prob(MICA) (see http://www.ncbi.nlm.nih.gov/pubmed/16776819), "Jiang" for 1 - difference between the sum of IC at pairs of terms and 2*IC at MICA (see http://arxiv.org/pdf/cmp-lg/9709008.pdf), "Pesquita" for graph information content similarity related to Tanimoto-Jacard index (ie. summed information content of common ancestors divided by summed information content of all ancestors of term1 and term2 (see http://www.ncbi.nlm.nih.gov/pubmed/18460186)). By default, it uses "Schlicker" method
`fast`	logical to indicate whether a vectorised fast computation is used. By default, it sets to true. It is always advisable to use this vectorised fast computation; since the conventional computation is just used for understanding scripts

parallel	logical to indicate whether parallel computation with multicores is used. By default, it sets to true, but not necessarily does so. Partly because parallel backends available will be system-specific (now only Linux or Mac OS). Also, it will depend on whether these two packages "foreach" and "doMC" have been installed. It can be installed via: `source("http://bioconductor.org/biocLite.R"); biocLite(c("foreach","doMC"))`. If not yet installed, this option will be disabled
multicores	an integer to specify how many cores will be registered as the multicore parallel backend to the 'foreach' package. If NULL, it will use a half of cores available in a user's computer. This option only works when parallel computation is enabled
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

It returns an object of class "igraph", with nodes for input terms and edges for pair-wise semantic similarity between terms.

Note

none

See Also

xDAGanno, xConverter

Examples

```
## Not run:
# 1) SNP-based ontology
# 1a) ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')
g

# 1b) load GWAS SNPs annotated by EF (an object of class "dgCMatrix" storing a spare matrix)
anno <- xRDataLoader(RData='GWAS2EF')

# 1c) prepare for ontology and its annotation information
dag <- xDAGanno(g=g, annotation=anno, path.mode="all_paths",
true.path.rule=TRUE, verbose=TRUE)

# 1d) calculate pair-wise semantic similarity between 5 randomly chosen terms
terms <- sample(V(dag)$name, 5)
sim <- xDAGsim(g=dag, terms=terms, method.term="Schlicker",
parallel=FALSE)
sim

############################################################
# 2) Gene-based ontology
# 2a) ig.MP (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.MP')

# 2b) load human genes annotated by MP (an object of class "GS" containing the 'gs' component)
GS <- xRDataLoader(RData='org.Hs.egMP')
anno <- GS$gs # notes: This is a list
```

```
# 2c) prepare for annotation data
dag <- xDAGanno(g=g, annotation=anno, path.mode="all_paths",
true.path.rule=TRUE, verbose=TRUE)

# 2d) calculate pair-wise semantic similarity between 5 randomly chosen terms
terms <- sample(V(dag)$name, 5)
sim <- xDAGsim(g=dag, terms=terms, method.term="Schlicker",
parallel=FALSE)
sim

## End(Not run)
```

---

xEnricher *Function to conduct enrichment analysis given the input data and the ontology and its annotation*

Description

xEnricher is supposed to conduct enrichment analysis given the input data and the ontology direct acyclic graph (DAG) and its annotation. It returns an object of class "eTerm". Enrichment analysis is based on either Fisher's exact test or Hypergeometric test. The test can respect the hierarchy of the ontology.

Usage

```
xEnricher(data, annotation, g, background = NULL, size.range = c(10,
2000),
min.overlap = 3, which.distance = NULL, test = c("hypergeo", "fisher",
"binomial"), p.adjust.method = c("BH", "BY", "bonferroni", "holm",
"hochberg", "hommel"), ontology.algorithm = c("none", "pc", "elim",
"lea"),
elim.pvalue = 0.01, lea.depth = 2, path.mode = c("all_paths",
"shortest_paths", "all_shortest_paths"), true.path.rule = TRUE,
verbose = T)
```

Arguments

data	an input vector containing a list of genes or SNPs of interest
annotation	the vertices/nodes for which annotation data are provided. It can be a sparse Matrix of class "dgCMatrix" (with variants/genes as rows and terms as columns), or a list of nodes/terms each containing annotation data, or an object of class 'GS' (basically a list for each node/term with annotation data)
g	an object of class "igraph" to represent DAG. It must have node/vertice attributes: "name" (i.e. "Term ID"), "term_id" (i.e. "Term ID"), "term_name" (i.e "Term Name") and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
background	a background vector. It contains a list of genes or SNPs as the test background. If NULL, by default all annotatable are used as background
size.range	the minimum and maximum size of members of each term in consideration. By default, it sets to a minimum of 10 but no more than 2000

`min.overlap` the minimum number of overlaps. Only those terms with members that overlap with input data at least min.overlap (3 by default) will be processed

`which.distance` which terms with the distance away from the ontology root (if any) is used to restrict terms in consideration. By default, it sets to 'NULL' to consider all distances

`test` the statistic test used. It can be "fisher" for using fisher's exact test, "hypergeo" for using hypergeometric test, or "binomial" for using binomial test. Fisher's exact test is to test the independence between gene group (genes belonging to a group or not) and gene annotation (genes annotated by a term or not), and thus compare sampling to the left part of background (after sampling without replacement). Hypergeometric test is to sample at random (without replacement) from the background containing annotated and non-annotated genes, and thus compare sampling to background. Unlike hypergeometric test, binomial test is to sample at random (with replacement) from the background with the constant probability. In terms of the ease of finding the significance, they are in order: hypergeometric test > binomial test > fisher's exact test. In other words, in terms of the calculated p-value, hypergeometric test < binomial test < fisher's exact test

`p.adjust.method` the method used to adjust p-values. It can be one of "BH", "BY", "bonferroni", "holm", "hochberg" and "hommel". The first two methods "BH" (widely used) and "BY" control the false discovery rate (FDR: the expected proportion of false discoveries amongst the rejected hypotheses); the last four methods "bonferroni", "holm", "hochberg" and "hommel" are designed to give strong control of the family-wise error rate (FWER). Notes: FDR is a less stringent condition than FWER

`ontology.algorithm` the algorithm used to account for the hierarchy of the ontology. It can be one of "none", "pc", "elim" and "lea". For details, please see 'Note' below

`elim.pvalue` the parameter only used when "ontology.algorithm" is "elim". It is used to control how to declare a signficantly enriched term (and subsequently all genes in this term are eliminated from all its ancestors)

`lea.depth` the parameter only used when "ontology.algorithm" is "lea". It is used to control how many maximum depth is used to consider the children of a term (and subsequently all genes in these children term are eliminated from the use for the recalculation of the signifance at this term)

`path.mode` the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)

`true.path.rule` logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true

`verbose` logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

an object of class "eTerm", a list with following components:

- `term_info`: a matrix of nTerm X 4 containing snp/gene set information, where nTerm is the number of terms, and the 4 columns are "id" (i.e. "Term ID"), "name" (i.e. "Term Name"), "namespace" and "distance"

- annotation: a list of terms containing annotations, each term storing its annotations. Always, terms are identified by "id"
- data: a vector containing input data in consideration. It is not always the same as the input data as only those mappable are retained
- background: a vector containing the background data. It is not always the same as the input data as only those mappable are retained
- overlap: a list of overlapped snp/gene sets, each storing snps/genes overlapped between a snp/gene set and the given input data (i.e. the snps/genes of interest). Always, gene sets are identified by "id"
- zscore: a vector containing z-scores
- pvalue: a vector containing p-values
- adjp: a vector containing adjusted p-values. It is the p value but after being adjusted for multiple comparisons
- call: the call that produced this result

Note

The interpretation of the algorithms used to account for the hierarchy of the ontology is:

- "none": does not consider the ontology hierarchy at all.
- "lea": computers the significance of a term in terms of the significance of its children at the maximum depth (e.g. 2). Precisely, once snps/genes are already annotated to any children terms with a more signficance than itself, then all these snps/genes are eliminated from the use for the recalculation of the signifance at that term. The final p-values takes the maximum of the original p-value and the recalculated p-value.
- "elim": computers the significance of a term in terms of the significance of its all children. Precisely, once snps/genes are already annotated to a signficantly enriched term under the cutoff of e.g. pvalue<1e-2, all these snps/genes are eliminated from the ancestors of that term).
- "pc": requires the significance of a term not only using the whole snps/genes as background but also using snps/genes annotated to all its direct parents/ancestors as background. The final p-value takes the maximum of both p-values in these two calculations.
- "Notes": the order of the number of significant terms is: "none" > "lea" > "elim" > "pc".

See Also

xDAGanno, xEnricherGenes, xEnricherSNPs

Examples

```
## Not run:
# 1) SNP-based enrichment analysis using GWAS Catalog traits (mapped to EF)
# 1a) ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')

# 1b) load GWAS SNPs annotated by EF (an object of class "dgCMatrix" storing a spare matrix)
anno <- xRDataLoader(RData='GWAS2EF')

# 1c) optionally, provide the test background (if not provided, all annotatable SNPs)
background <- rownames(anno)
```

```
# 1d) provide the input SNPs of interest (eg 'EFO:0002690' for 'systemic lupus erythematosus')
ind <- which(colnames(anno)=='EFO:0002690')
data <- rownames(anno)[anno[,ind]==1]
data

# 1e) perform enrichment analysis
eTerm <- xEnricher(data=data, annotation=anno, background=background,
g=g, path.mode=c("all_paths"))

# 1f) view enrichment results for the top significant terms
xEnrichViewer(eTerm)

# 1f') save enrichment results to the file called 'EF_enrichments.txt'
res <- xEnrichViewer(eTerm, top_num=length(eTerm$adjp), sortBy="adjp",
details=TRUE)
output <- data.frame(term=rownames(res), res)
utils::write.table(output, file="EF_enrichments.txt", sep="\t",
row.names=FALSE)

# 1g) visualise the top 10 significant terms in the ontology hierarchy
g <- xRDataLoader(RData='ig.EF')
g
nodes_query <- names(sort(eTerm$adjp)[1:10])
nodes.highlight <- rep("red", length(nodes_query))
names(nodes.highlight) <- nodes_query
subg <- dnet::dDAGinduce(g, nodes_query)
# color-code terms according to the adjust p-values (taking the form of 10-based negative logarithm)
dnet::visDAG(g=subg, data=-1*log10(eTerm$adjp[V(subg)$name]),
node.info="both", zlim=c(0,2), node.attrs=list(color=nodes.highlight))
# color-code terms according to the z-scores
dnet::visDAG(g=subg, data=eTerm$zscore[V(subg)$name], node.info="both",
colormap="darkblue-white-darkorange",
node.attrs=list(color=nodes.highlight))

## End(Not run)
```

`xEnricherGenes` *Function to conduct enrichment analysis given a list of genes and the ontology in query*

Description

`xEnricherGenes` is supposed to conduct enrichment analysis given the input data and the ontology in query. It returns an object of class "eTerm". Enrichment analysis is based on either Fisher's exact test or Hypergeometric test. The test can respect the hierarchy of the ontology. Now it supports enrichment analysis using a wide variety of ontologies such as Gene Ontology and Phenotype Ontologies.

Usage

```
xEnricherGenes(data, background = NULL, ontology = c("GOBP", "GOMF",
"GOCC",
"PS", "PS2", "SF", "DO", "HPPA", "HPMI", "HPCM", "HPMA", "MP",
"MsigdbH",
```

```
"MsigdbC1", "MsigdbC2CGP", "MsigdbC2CP", "MsigdbC2KEGG",
"MsigdbC2REACTOME",
"MsigdbC2BIOCARTA", "MsigdbC3TFT", "MsigdbC3MIR", "MsigdbC4CGN",
"MsigdbC4CM",
"MsigdbC5BP", "MsigdbC5MF", "MsigdbC5CC", "MsigdbC6", "MsigdbC7",
"DGIdb"),
size.range = c(10, 2000), min.overlap = 3, which.distance = NULL,
test = c("hypergeo", "fisher", "binomial"), p.adjust.method = c("BH",
"BY", "bonferroni", "holm", "hochberg", "hommel"),
ontology.algorithm = c("none", "pc", "elim", "lea"), elim.pvalue =
0.01,
lea.depth = 2, path.mode = c("all_paths", "shortest_paths",
"all_shortest_paths"), true.path.rule = F, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

data	an input vector. It contains a list of Gene Symbols of interest
background	a background vector. It contains a list of Gene Symbols as the test background. If NULL, by default all annotatable are used as background
ontology	the ontology supported currently. It can be "GOBP" for Gene Ontology Biological Process, "GOMF" for Gene Ontology Molecular Function, "GOCC" for Gene Ontology Cellular Component, "PS" for phylostratific age information, "PS2" for the collapsed PS version (inferred ancestors being collapsed into one with the known taxonomy information), "SF" for domain superfamily assignments, "DO" for Disease Ontology, "HPPA" for Human Phenotype Phenotypic Abnormality, "HPMI" for Human Phenotype Mode of Inheritance, "HPCM" for Human Phenotype Clinical Modifier, "HPMA" for Human Phenotype Mortality Aging, "MP" for Mammalian Phenotype, and Drug-Gene Interaction database (DGIdb) for drugable categories, and the molecular signatures database (Msigdb, including "MsigdbH", "MsigdbC1", "MsigdbC2CGP", "MsigdbC2CP", "MsigdbC2KEGG", "MsigdbC2REACTOME", "MsigdbC2BIOCARTA", "MsigdbC3TFT", "MsigdbC3MIR", "MsigdbC4CGN", "MsigdbC4CM", "MsigdbC5BP", "MsigdbC5MF", "MsigdbC5CC", "MsigdbC6", "MsigdbC7")
size.range	the minimum and maximum size of members of each term in consideration. By default, it sets to a minimum of 10 but no more than 2000
min.overlap	the minimum number of overlaps. Only those terms with members that overlap with input data at least min.overlap (3 by default) will be processed
which.distance	which terms with the distance away from the ontology root (if any) is used to restrict terms in consideration. By default, it sets to 'NULL' to consider all distances
test	the statistic test used. It can be "fisher" for using fisher's exact test, "hypergeo" for using hypergeometric test, or "binomial" for using binomial test. Fisher's exact test is to test the independence between gene group (genes belonging to a group or not) and gene annotation (genes annotated by a term or not), and thus compare sampling to the left part of background (after sampling without replacement). Hypergeometric test is to sample at random (without replacement) from the background containing annotated and non-annotated genes, and thus compare sampling to background. Unlike hypergeometric test, binomial test is to sample at random (with replacement) from the background with the

constant probability. In terms of the ease of finding the significance, they are in order: hypergeometric test > binomial test > fisher's exact test. In other words, in terms of the calculated p-value, hypergeometric test < binomial test < fisher's exact test

`p.adjust.method`
: the method used to adjust p-values. It can be one of "BH", "BY", "bonferroni", "holm", "hochberg" and "hommel". The first two methods "BH" (widely used) and "BY" control the false discovery rate (FDR: the expected proportion of false discoveries amongst the rejected hypotheses); the last four methods "bonferroni", "holm", "hochberg" and "hommel" are designed to give strong control of the family-wise error rate (FWER). Notes: FDR is a less stringent condition than FWER

`ontology.algorithm`
: the algorithm used to account for the hierarchy of the ontology. It can be one of "none", "pc", "elim" and "lea". For details, please see 'Note' below

`elim.pvalue`
: the parameter only used when "ontology.algorithm" is "elim". It is used to control how to declare a signficantly enriched term (and subsequently all genes in this term are eliminated from all its ancestors)

`lea.depth`
: the parameter only used when "ontology.algorithm" is "lea". It is used to control how many maximum depth is used to consider the children of a term (and subsequently all genes in these children term are eliminated from the use for the recalculation of the signifance at this term)

`path.mode`
: the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)

`true.path.rule`
: logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to false

`verbose`
: logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display

`RData.location`
: the characters to tell the location of built-in RData files. See `xRDataLoader` for details

Value

an object of class "eTerm", a list with following components:

- `term_info`: a matrix of nTerm X 4 containing snp/gene set information, where nTerm is the number of terms, and the 4 columns are "id" (i.e. "Term ID"), "name" (i.e. "Term Name"), "namespace" and "distance"
- `annotation`: a list of terms containing annotations, each term storing its annotations. Always, terms are identified by "id"
- `data`: a vector containing input data in consideration. It is not always the same as the input data as only those mappable are retained
- `background`: a vector containing the background data. It is not always the same as the input data as only those mappable are retained
- `overlap`: a list of overlapped snp/gene sets, each storing snps overlapped between a snp/gene set and the given input data (i.e. the snps of interest). Always, gene sets are identified by "id"
- `zscore`: a vector containing z-scores

- pvalue: a vector containing p-values
- adjp: a vector containing adjusted p-values. It is the p value but after being adjusted for multiple comparisons
- call: the call that produced this result

Note

The interpretation of the algorithms used to account for the hierarchy of the ontology is:

- "none": does not consider the ontology hierarchy at all.
- "lea": computers the significance of a term in terms of the significance of its children at the maximum depth (e.g. 2). Precisely, once snps are already annotated to any children terms with a more signficance than itself, then all these snps are eliminated from the use for the recalculation of the signifance at that term. The final p-values takes the maximum of the original p-value and the recalculated p-value.
- "elim": computers the significance of a term in terms of the significance of its all children. Precisely, once snps are already annotated to a signficantly enriched term under the cutoff of e.g. pvalue<1e-2, all these snps are eliminated from the ancestors of that term).
- "pc": requires the significance of a term not only using the whole snps as background but also using snps annotated to all its direct parents/ancestors as background. The final p-value takes the maximum of both p-values in these two calculations.
- "Notes": the order of the number of significant terms is: "none" > "lea" > "elim" > "pc".

See Also

xRDataLoader, xEnricher

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)

# Gene-based enrichment analysis using Mammalian Phenotype Ontology (MP)
# a) provide the input Genes of interest (eg 100 randomly chosen human genes)
## load human genes
org.Hs.eg <- xRDataLoader(RData='org.Hs.eg')
data <- as.character(sample(org.Hs.eg$gene_info$Symbol, 100))
data

# optionally, provide the test background (if not provided, all human genes)
#background <- as.character(org.Hs.eg$gene_info$Symbol)

# b) perform enrichment analysis
eTerm <- xEnricherGenes(data=data, ontology="MP")

# c) view enrichment results for the top significant terms
xEnrichViewer(eTerm)

# d) save enrichment results to the file called 'MP_enrichments.txt'
res <- xEnrichViewer(eTerm, top_num=length(eTerm$adjp), sortBy="adjp",
details=TRUE)
output <- data.frame(term=rownames(res), res)
```

```
utils::write.table(output, file="MP_enrichments.txt", sep="\t",
row.names=FALSE)

# e) visualise the top 10 significant terms in the ontology hierarchy
## load ig.MP (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader(RData='ig.MP')
g
nodes_query <- names(sort(eTerm$adjp)[1:10])
nodes.highlight <- rep("red", length(nodes_query))
names(nodes.highlight) <- nodes_query
subg <- dnet::dDAGinduce(g, nodes_query)
# color-code terms according to the adjust p-values (taking the form of 10-based negative logarithm)
dnet::visDAG(g=subg, data=-1*log10(eTerm$adjp[V(subg)$name]),
node.info="both", zlim=c(0,2), node.attrs=list(color=nodes.highlight))
# color-code terms according to the z-scores
dnet::visDAG(g=subg, data=eTerm$zscore[V(subg)$name], node.info="both",
colormap="darkblue-white-darkorange",
node.attrs=list(color=nodes.highlight))

## End(Not run)
```

`xEnricherSNPs` *Function to conduct enrichment analysis given a list of SNPs and the ontology in query*

Description

`xEnricherSNPs` is supposed to conduct enrichment analysis given the input data and the ontology in query. It returns an object of class "eTerm". Enrichment analysis is based on either Fisher's exact test or Hypergeometric test. The test can respect the hierarchy of the ontology. Now it supports enrichment analysis for SNPs using GWAS Catalog traits mapped to Experimental Factor Ontology. If required, additional SNPs that are in linkage disequilibrium (LD) with input SNPs are also be used for test.

Usage

```
xEnricherSNPs(data, background = NULL, ontology = c("EF", "EF_disease",
"EF_phenotype", "EF_bp"), include.LD = NA, LD.r2 = 0.8,
size.range = c(10, 2000), min.overlap = 3, which.distance = NULL,
test = c("hypergeo", "fisher", "binomial"), p.adjust.method = c("BH",
"BY", "bonferroni", "holm", "hochberg", "hommel"),
ontology.algorithm = c("none", "pc", "elim", "lea"), elim.pvalue =
0.01,
lea.depth = 2, path.mode = c("all_paths", "shortest_paths",
"all_shortest_paths"), true.path.rule = T, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

`data`	an input vector. It contains a list of SNPs of interest
`background`	a background vector. It contains a list of SNPs as the test background. If NULL, by default all annotatable are used as background

`ontology` the ontology supported currently. Now it is only "EF" for Experimental Factor Ontology (used to annotate GWAS Catalog SNPs). However, there are several subparts of this ontology to choose: 'EF_disease' for the subpart under the term 'disease' (EFO:0000408), 'EF_phenotype' for the subpart under the term 'phenotype' (EFO:0000651), 'EF_bp' for the subpart under the term 'biological process' (GO:0008150)

`include.LD` additional SNPs in LD with Lead SNPs are also included. By default, it is 'NA' to disable this option. Otherwise, LD SNPs will be included based on one or more of 26 populations and 5 super populations from 1000 Genomics Project data (phase 3). The population can be one of 5 super populations ("AFR", "AMR", "EAS", "EUR", "SAS"), or one of 26 populations ("ACB", "ASW", "BEB", "CDX", "CEU", "CHB", "CHS", "CLM", "ESN", "FIN", "GBR", "GIH", "GWD", "IBS", "ITU", "JPT", "KHV", "LWK", "MSL", "MXL", "PEL", "PJL", "PUR", "STU", "TSI", "YRI"). Explanations for population code can be found at http://www.1000genomes.org/faq/which-populations-are-part-your-study

`LD.r2` the LD r2 value. By default, it is 0.8, meaning that SNPs in LD (r2>=0.8) with input SNPs will be considered as LD SNPs. It can be any value from 0.8 to 1

`size.range` the minimum and maximum size of members of each term in consideration. By default, it sets to a minimum of 10 but no more than 2000

`min.overlap` the minimum number of overlaps. Only those terms with members that overlap with input data at least min.overlap (3 by default) will be processed

`which.distance` which terms with the distance away from the ontology root (if any) is used to restrict terms in consideration. By default, it sets to 'NULL' to consider all distances

`test` the statistic test used. It can be "fisher" for using fisher's exact test, "hypergeo" for using hypergeometric test, or "binomial" for using binomial test. Fisher's exact test is to test the independence between gene group (genes belonging to a group or not) and gene annotation (genes annotated by a term or not), and thus compare sampling to the left part of background (after sampling without replacement). Hypergeometric test is to sample at random (without replacement) from the background containing annotated and non-annotated genes, and thus compare sampling to background. Unlike hypergeometric test, binomial test is to sample at random (with replacement) from the background with the constant probability. In terms of the ease of finding the significance, they are in order: hypergeometric test > binomial test > fisher's exact test. In other words, in terms of the calculated p-value, hypergeometric test < binomial test < fisher's exact test

`p.adjust.method` the method used to adjust p-values. It can be one of "BH", "BY", "bonferroni", "holm", "hochberg" and "hommel". The first two methods "BH" (widely used) and "BY" control the false discovery rate (FDR: the expected proportion of false discoveries amongst the rejected hypotheses); the last four methods "bonferroni", "holm", "hochberg" and "hommel" are designed to give strong control of the family-wise error rate (FWER). Notes: FDR is a less stringent condition than FWER

`ontology.algorithm` the algorithm used to account for the hierarchy of the ontology. It can be one of "none", "pc", "elim" and "lea". For details, please see 'Note' below

`elim.pvalue` the parameter only used when "ontology.algorithm" is "elim". It is used to control how to declare a signficantly enriched term (and subsequently all genes in this term are eliminated from all its ancestors)

lea.depth	the parameter only used when "ontology.algorithm" is "lea". It is used to control how many maximum depth is used to consider the children of a term (and subsequently all genes in these children term are eliminated from the use for the recalculation of the signifance at this term)
path.mode	the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)
true.path.rule	logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display
RData.location	the characters to tell the location of built-in RData files. See xRDataLoader for details

Value

an object of class "eTerm", a list with following components:

- term_info: a matrix of nTerm X 4 containing snp/gene set information, where nTerm is the number of terms, and the 4 columns are "id" (i.e. "Term ID"), "name" (i.e. "Term Name"), "namespace" and "distance"
- annotation: a list of terms containing annotations, each term storing its annotations. Always, terms are identified by "id"
- data: a vector containing input data in consideration. It is not always the same as the input data as only those mappable are retained
- background: a vector containing the background data. It is not always the same as the input data as only those mappable are retained
- overlap: a list of overlapped snp/gene sets, each storing snps overlapped between a snp/gene set and the given input data (i.e. the snps of interest). Always, gene sets are identified by "id"
- zscore: a vector containing z-scores
- pvalue: a vector containing p-values
- adjp: a vector containing adjusted p-values. It is the p value but after being adjusted for multiple comparisons
- call: the call that produced this result

Note

The interpretation of the algorithms used to account for the hierarchy of the ontology is:

- "none": does not consider the ontology hierarchy at all.
- "lea": computers the significance of a term in terms of the significance of its children at the maximum depth (e.g. 2). Precisely, once snps are already annotated to any children terms with a more signficance than itself, then all these snps are eliminated from the use for the recalculation of the signifance at that term. The final p-values takes the maximum of the original p-value and the recalculated p-value.
- "elim": computers the significance of a term in terms of the significance of its all children. Precisely, once snps are already annotated to a signficantly enriched term under the cutoff of e.g. pvalue<1e-2, all these snps are eliminated from the ancestors of that term).

- "pc": requires the significance of a term not only using the whole snps as background but also using snps annotated to all its direct parents/ancestors as background. The final p-value takes the maximum of both p-values in these two calculations.
- "Notes": the order of the number of significant terms is: "none" > "lea" > "elim" > "pc".

See Also

xRDataLoader, xEnricher

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)

# SNP-based enrichment analysis using GWAS Catalog traits (mapped to EF)
# a) provide the input SNPs of interest (eg 'EFO:0002690' for 'systemic lupus erythematosus')
## load GWAS SNPs annotated by EF (an object of class "dgCMatrix" storing a spare matrix)
anno <- xRDataLoader(RData='GWAS2EF')
ind <- which(colnames(anno)=='EFO:0002690')
data <- rownames(anno)[anno[,ind]==1]
data

# optionally, provide the test background (if not provided, all annotatable SNPs)
#background <- rownames(anno)

# b) perform enrichment analysis
eTerm <- xEnricherSNPs(data=data, ontology="EF",
path.mode=c("all_paths"))

# b') optionally, enrichment analysis for input SNPs plus additional SNPs that are in LD with input SNPs
## LD based on European population (EUR) with r2>=0.8
#eTerm <- xEnricherSNPs(data=data, include.LD="EUR", LD.r2=0.8)

# c) view enrichment results for the top significant terms
xEnrichViewer(eTerm)

# d) save enrichment results to the file called 'EF_enrichments.txt'
res <- xEnrichViewer(eTerm, top_num=length(eTerm$adjp), sortBy="adjp",
details=TRUE)
output <- data.frame(term=rownames(res), res)
utils::write.table(output, file="EF_enrichments.txt", sep="\t",
row.names=FALSE)

# e) visualise the top 10 significant terms in the ontology hierarchy
## load ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')
g
nodes_query <- names(sort(eTerm$adjp)[1:10])
nodes.highlight <- rep("red", length(nodes_query))
names(nodes.highlight) <- nodes_query
subg <- dnet::dDAGinduce(g, nodes_query)
# color-code terms according to the adjust p-values (taking the form of 10-based negative logarithm)
dnet::visDAG(g=subg, data=-1*log10(eTerm$adjp[V(subg)$name]),
node.info="both", zlim=c(0,2), node.attrs=list(color=nodes.highlight))
```

```
# color-code terms according to the z-scores
dnet::visDAG(g=subg, data=eTerm$zscore[V(subg)$name], node.info="both",
colormap="darkblue-white-darkorange",
node.attrs=list(color=nodes.highlight))

## End(Not run)
```

---

xEnricherYours *Function to conduct enrichment analysis given YOUR own input data*

Description

xEnricherYours is supposed to conduct enrichment analysis given the input data and the ontology in query. It returns an object of class "eTerm". Enrichment analysis is based on either Fisher's exact test or Hypergeometric test.

Usage

```
xEnricherYours(data.file, annotation.file, background.file = NULL,
size.range = c(10, 2000), min.overlap = 3, test = c("hypergeo",
"fisher", "binomial"), p.adjust.method = c("BH", "BY", "bonferroni",
"holm",
"hochberg", "hommel"), verbose = T)
```

Arguments

data.file	an input data file, containing a list of entities (e.g. genes or SNPs) to test. The entities can be anything, for example, in this file http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt, the entities are InterPro domains (InterPro). As seen in this example, entries in the first column must be domains. If the file also contains other columns, these additional columns will be ignored. Alternatively, the data.file can be a matrix or data frame, assuming that input file has been read. Note: the file should use the tab delimiter as the field separator between columns
annotation.file	an input annotation file containing annotations between entities and ontology terms. For example, a file containing annotations between InterPro domains and GO Molecular Function (GOMF) terms can be found in http://dcgor.r-forge.r-project.org/data/InterPro/Domain2GOMF.txt. As seen in this example, the input file must contain two columns: 1st column for domains, 2nd column for ontology terms. If there are additional columns, these columns will be ignored. Alternatively, the annotation.file can be a matrix or data frame, assuming that input file has been read. Note: the file should use the tab delimiter as the field separator between columns
background.file	an input background file containing a list of entities as the test background. The file format is the same as 'data.file'. By default, it is NULL meaning all annotatable entities (i.e. those entities in 'annotation.file') are used as background
size.range	the minimum and maximum size of members of each term in consideration. By default, it sets to a minimum of 10 but no more than 2000

`min.overlap` the minimum number of overlaps. Only those terms with members that overlap with input data at least min.overlap (3 by default) will be processed

`test` the statistic test used. It can be "fisher" for using fisher's exact test, "hypergeo" for using hypergeometric test, or "binomial" for using binomial test. Fisher's exact test is to test the independence between gene group (genes belonging to a group or not) and gene annotation (genes annotated by a term or not), and thus compare sampling to the left part of background (after sampling without replacement). Hypergeometric test is to sample at random (without replacement) from the background containing annotated and non-annotated genes, and thus compare sampling to background. Unlike hypergeometric test, binomial test is to sample at random (with replacement) from the background with the constant probability. In terms of the ease of finding the significance, they are in order: hypergeometric test > binomial test > fisher's exact test. In other words, in terms of the calculated p-value, hypergeometric test < binomial test < fisher's exact test

`p.adjust.method` the method used to adjust p-values. It can be one of "BH", "BY", "bonferroni", "holm", "hochberg" and "hommel". The first two methods "BH" (widely used) and "BY" control the false discovery rate (FDR: the expected proportion of false discoveries amongst the rejected hypotheses); the last four methods "bonferroni", "holm", "hochberg" and "hommel" are designed to give strong control of the family-wise error rate (FWER). Notes: FDR is a less stringent condition than FWER

`verbose` logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display

Value

an object of class "eTerm", a list with following components:

- `term_info`: a matrix of nTerm X 4 containing snp/gene set information, where nTerm is the number of terms, and the 4 columns are "id" (i.e. "Term ID"), "name" (i.e. "Term Name"), "namespace" and "distance"
- `annotation`: a list of terms containing annotations, each term storing its annotations. Always, terms are identified by "id"
- `data`: a vector containing input data in consideration. It is not always the same as the input data as only those mappable are retained
- `background`: a vector containing the background data. It is not always the same as the input data as only those mappable are retained
- `overlap`: a list of overlapped snp/gene sets, each storing snps overlapped between a snp/gene set and the given input data (i.e. the snps of interest). Always, gene sets are identified by "id"
- `zscore`: a vector containing z-scores
- `pvalue`: a vector containing p-values
- `adjp`: a vector containing adjusted p-values. It is the p value but after being adjusted for multiple comparisons
- `call`: the call that produced this result

Note

None

See Also

xEnricher

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)

# Enrichment analysis using your own data
# a) provide your own data (i.e. InterPro domains and their annotations by GO terms)
## All InterPro domains
input.file <-
"http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt"
data <- utils::read.delim(input.file, header=F, row.names=NULL,
stringsAsFactors=F)[,1]
## provide the input domains of interest (eg 100 randomly chosen domains)
data.file <- sample(data, 100)
## InterPro domains annotated by GO Molecular Function (GOMF) terms
annotation.file <-
"http://dcgor.r-forge.r-project.org/data/InterPro/Domain2GOMF.txt"

# b) perform enrichment analysis
eTerm <- xEnricherYours(data.file=data.file,
annotation.file=annotation.file)

# c) view enrichment results for the top significant terms
xEnrichViewer(eTerm)

# d) save enrichment results to the file called 'Yours_enrichments.txt'
output <- xEnrichViewer(eTerm, top_num=length(eTerm$adjp),
sortBy="adjp", details=TRUE)
utils::write.table(output, file="Yours_enrichments.txt", sep="\t",
row.names=FALSE)

## End(Not run)
```

---

xEnrichViewer *Function to view enrichment results*

Description

xEnrichViewer is supposed to view results of enrichment analysis.

Usage

```
xEnrichViewer(eTerm, top_num = 10, sortBy = c("adjp", "pvalue",
"zscore",
"nAnno", "nOverlap", "none"), decreasing = NULL, details = F)
```

Arguments

eTerm	an object of class "eTerm"
top_num	the number of the top terms (sorted according to 'sortBy' below) will be viewed
sortBy	which statistics will be used for sorting and viewing gene sets (terms). It can be "adjp" for adjusted p value, "pvalue" for p value, "zscore" for enrichment z-score, "nAnno" for the number of sets (terms), "nOverlap" for the number in overlaps, and "none" for ordering according to ID of terms
decreasing	logical to indicate whether to sort in a decreasing order. If it is null, it would be true for "zscore", "nAnno" or "nOverlap"; otherwise it would be false
details	logical to indicate whether the detailed information of gene sets (terms) is also viewed. By default, it sets to false for no inclusion

Value

a data frame with following components:

- id: term ID; as rownames
- name: term name
- nAnno: number in members annotated by a term
- nOverlap: number in overlaps
- zscore: enrichment z-score
- pvalue: nominal p value
- adjp: adjusted p value
- distance: term distance; optional, it is only appended when "details" is true
- members: members (represented as Gene Symbols) in overlaps; optional, it is only appended when "details" is true

Note

none

See Also

xEnricherGenes, xEnricherSNPs

Examples

```
## Not run:
xEnrichViewer(eTerm)

## End(Not run)
```

xFunArgs *Function to assign (and evaluate) arguments with default values for a given function*

Description

xFunArgs is supposed to assign (and evaluate) arguments with default values for a given function.

Usage

```
xFunArgs(fun, action = F, verbose = TRUE)
```

Arguments

fun	character specifying the name of the function
action	logical to indicate whether the function will act as it should be (with assigned values in the current environment). By default, it sets to FALSE
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display

Value

a list containing arguments and their default values

Note

This function is potentially useful when debugging as it frees developers from specifying default values for all arguments except those arguments of interest

See Also

xFunArgs

Examples

```
fun <- "xRDataLoader"
xFunArgs(fun)
```

xRd2HTML *Function to convert Rd files to HTML files*

Description

xRd2HTML is supposed to convert Rd files to HTML files.

Usage

```
xRd2HTML(path.from = "./XGR/man", path.to = "./XGR/vignettes")
```

Arguments

path.from	a directory containing Rd files converted from
path.to	a directory containing HTML files converted to

Value

none

Note

This auxiliary function helps create a new package.

See Also

xRd2HTML

Examples

```
# xRd2HTML(path.from="./XGR/man", path.to="./XGR/vignettes")
```

---

xRDataLoader *Function to load the package built-in RData*

Description

xRDataLoader is supposed to load the package built-in RData.

Usage

```
xRDataLoader(RData = c(NA, "GWAS2EF", "GWAS_LD", "IlluminaHumanHT",
"IlluminaOmniExpress", "ig.DO", "ig.EF", "ig.GOBP", "ig.GOCC",
"ig.GOMF",
"ig.HPCM", "ig.HPMA", "ig.HPMI", "ig.HPPA", "ig.MP", "org.Hs.eg",
"org.Hs.egDGIdb", "org.Hs.egDO", "org.Hs.egGOBP", "org.Hs.egGOCC",
"org.Hs.egGOMF", "org.Hs.egHPCM", "org.Hs.egHPMA", "org.Hs.egHPMI",
"org.Hs.egHPPA", "org.Hs.egMP", "org.Hs.egMsigdbC1",
"org.Hs.egMsigdbC2BIOCARTA", "org.Hs.egMsigdbC2CGP",
"org.Hs.egMsigdbC2CP",
"org.Hs.egMsigdbC2KEGG", "org.Hs.egMsigdbC2REACTOME",
"org.Hs.egMsigdbC3MIR", "org.Hs.egMsigdbC3TFT", "org.Hs.egMsigdbC4CGN",
"org.Hs.egMsigdbC4CM", "org.Hs.egMsigdbC5BP", "org.Hs.egMsigdbC5CC",
"org.Hs.egMsigdbC5MF", "org.Hs.egMsigdbC6", "org.Hs.egMsigdbC7",
"org.Hs.egMsigdbH", "org.Hs.egPS", "org.Hs.egSF", "org.Hs.string",
"org.Hs.PCommons_DN", "org.Hs.PCommons_UN"), RData.customised = NULL,
verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

RData	which built-in RData to load. It can be one of "GWAS2EF", "GWAS_LD", "IlluminaHumanHT", "IlluminaOmniExpress", "ig.DO", "ig.EF", "ig.GOBP", "ig.GOCC", "ig.GOMF", "ig.HPCM", "ig.HPMA", "ig.HPMI", "ig.HPPA", "ig.MP", "org.Hs.eg", "org.Hs.egDGIdb", "org.Hs.egDO", "org.Hs.egGOBP", "org.Hs.egGOCC", "org.Hs.egGOMF", "org.Hs.egHPCM", "org.Hs.egHPMA", "org.Hs.egHPMI", "org.Hs.egHPPA", "org.Hs.egMP", "org.Hs.egMsigdbC1", "org.Hs.egMsigdbC2BIOCARTA", "org.Hs.egMsigdbC2CGP", "org.Hs.egMsigdbC2CP", "org.Hs.egMsigdbC2KEGG", "org.Hs.egMsigdbC2REACTOME", "org.Hs.egMsigdbC3MIR", "org.Hs.egMsigdbC3TFT", "org.Hs.egMsigdbC4CGN", "org.Hs.egMsigdbC4CM", "org.Hs.egMsigdbC5BP", "org.Hs.egMsigdbC5CC", "org.Hs.egMsigdbC5MF", "org.Hs.egMsigdbC6", "org.Hs.egMsigdbC7", "org.Hs.egMsigdbH", "org.Hs.egPS", "org.Hs.egSF", "org.Hs.string", "org.Hs.PCommons_DN", "org.Hs.PCommons_UN"
RData.customised	a file name for RData-formatted file. By default, it is NULL. It is designed when the user wants to import customised RData that are not listed in the above argument 'RData'. However, this argument can be always used even for those RData that are listed in the argument 'RData'
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display
RData.location	the characters to tell the location of built-in RData files. By default, it remotely locates at https://github.com/hfang-bristol/RDataCentre/blob/master/XGR. For the user equipped with fast internet connection, this option can be just left as default. But it is always advisable to download these files locally. Especially when the user needs to run this function many times, there is no need to ask the function to remotely download every time (also it will unnecessarily increase the runtime). For examples, these files (as a whole or part of them) can be first downloaded into your current working directory, and then set this option as: $RData.location = "."$. Surely, the location can be anywhere as long as the user provides the correct path pointing to (otherwise, the script will have to remotely download each time)

Value

any use-specified variable that is given on the right side of the assigement sign '<-', which contains the loaded RData.

Note

If there are no use-specified variable that is given on the right side of the assigement sign '<-', then no RData will be loaded onto the working environment.

See Also

xRDataLoader

Examples

```
## Not run:
org.Hs.eg <- xRDataLoader(RData='org.Hs.eg')
ig.HPPA <- xRDataLoader(RData='ig.HPPA')
org.Hs.egHPPA <- xRDataLoader(RData='org.Hs.egHPPA')
```

```
org.Hs.egHPPA <- xRDataLoader(RData.customised='org.Hs.egHPPA')

## End(Not run)
```

---

xRdWrap *Function to wrap texts from Rd files*

Description

xRdWrap is supposed to wrap texts from Rd files under a given directory.

Usage

```
xRdWrap(path = "./XGR/man", remove.dontrun = FALSE)
```

Arguments

path	a directory containing Rd files
remove.dontrun	logical to indicate whether to remove the restriction of not running examples. By default, it sets to FALSE without any modefications

Value

none

Note

This auxiliary function helps create a new package. The orignal Rd files will be replaced with new ones.

See Also

xRdWrap

Examples

```
# xRdWrap(path="./XGR/man", remove.dontrun=FALSE)
```

`xSocialiser` *Function to calculate pair-wise semantic similarity given the input data and the ontology and its annotation*

Description

`xSocialiser` is supposed to calculate pair-wise semantic similarity given the input data and the ontology direct acyclic graph (DAG) and its annotation. It returns an object of class "igraph", a network representation of socialized genes/SNPs. It first calculates semantic similarity between terms and then derives semantic similarity from term-term semantic similarity. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
xSocialiser(data, annotation, g, measure = c("BM.average", "BM.max",
"BM.complete", "average", "max"), method.term = c("Resnik", "Lin",
"Schlicker", "Jiang", "Pesquita"), rescale = TRUE, force = TRUE,
fast = TRUE, parallel = TRUE, multicores = NULL,
path.mode = c("all_paths", "shortest_paths", "all_shortest_paths"),
true.path.rule = TRUE, verbose = T)
```

Arguments

`data`	an input vector containing a list of genes or SNPs of interest between which pair-wise semantic similarity is calculated/socialized. If NULL, all annotatable will be used for calcluation, which is very prohibitively expensive!
`annotation`	the vertices/nodes for which annotation data are provided. It can be a sparse Matrix of class "dgCMatrix" (with variants/genes as rows and terms as columns), or a list of nodes/terms each containing annotation data, or an object of class 'GS' (basically a list for each node/term with annotation data)
`g`	an object of class "igraph" to represent DAG. It must have node/vertice attributes: "name" (i.e. "Term ID"), "term_id" (i.e. "Term ID"), "term_name" (i.e "Term Name") and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
`measure`	the measure used to derive semantic similarity between genes/SNPs from semantic similarity between terms. Take the semantic similartity between SNPs as an example. It can be "average" for average similarity between any two terms (one from SNP 1, the other from SNP 2), "max" for the maximum similarity between any two terms, "BM.average" for best-matching (BM) based average similarity (i.e. for each term of either SNP, first calculate maximum similarity to any term in the other SNP, then take average of maximum similarity; the final BM-based average similiary is the pre-calculated average between two SNPs in pair), "BM.max" for BM based maximum similarity (i.e. the same as "BM.average", but the final BM-based maximum similiary is the maximum of the pre-calculated average between two SNPs in pair), "BM.complete" for BM-based complete-linkage similarity (inspired by complete-linkage concept: the least of any maximum similarity between a term of one SNP and a term of the other SNP). When comparing BM-based similarity between SNPs, "BM.average" and "BM.max" are sensitive to the number of terms invovled; instead, "BM.complete" is much robust in this aspect. By default, it uses "BM.average"

`method.term` the method used to measure semantic similarity between terms. It can be "Resnik" for information content (IC) of most informative common ancestor (MICA) (see http://arxiv.org/pdf/cmp-lg/9511007.pdf), "Lin" for 2*IC at MICA divided by the sum of IC at pairs of terms (see https://www.cse.iitb.ac.in/~cs626-449/Papers/WordSimilarity/3.pdf), "Schlicker" for weighted version of 'Lin' by the 1-prob(MICA) (see http://www.ncbi.nlm.nih.gov/pubmed/16776819), "Jiang" for 1 - difference between the sum of IC at pairs of terms and 2*IC at MICA (see http://arxiv.org/pdf/cmp-lg/9709008.pdf), "Pesquita" for graph information content similarity related to Tanimoto-Jacard index (ie. summed information content of common ancestors divided by summed information content of all ancestors of term1 and term2 (see http://www.ncbi.nlm.nih.gov/pubmed/18460186))

`rescale` logical to indicate whether the resulting values are rescaled to the range [0,1]. By default, it sets to true

`force` logical to indicate whether the only most specific terms (for each SNP) will be used. By default, it sets to true. It is always advisable to use this since it is computationally fast but without compromising accuracy (considering the fact that true-path-rule has been applied when running xDAGanno)

`fast` logical to indicate whether a vectorised fast computation is used. By default, it sets to true. It is always advisable to use this vectorised fast computation; since the conventional computation is just used for understanding scripts

`parallel` logical to indicate whether parallel computation with multicores is used. By default, it sets to true, but not necessarily does so. Partly because parallel backends available will be system-specific (now only Linux or Mac OS). Also, it will depend on whether these two packages "foreach" and "doMC" have been installed. It can be installed via: `source("http://bioconductor.org/biocLite.R"); biocLite(c("foreach","doMC"))`. If not yet installed, this option will be disabled

`multicores` an integer to specify how many cores will be registered as the multicore parallel backend to the 'foreach' package. If NULL, it will use a half of cores available in a user's computer. This option only works when parallel computation is enabled

`path.mode` the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)

`true.path.rule` logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true

`verbose` logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

It returns an object of class "igraph", with nodes for input genes/SNPs and edges for pair-wise semantic similarity between them.

Note

For the mode "shortest_paths", the induced subgraph is the most concise, and thus informative for visualisation when there are many nodes in query, while the mode "all_paths" results in the complete subgraph.

See Also

xDAGsim, xSocialiserGenes, xSocialiserSNPs

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)

# 1) SNP-based enrichment analysis using GWAS Catalog traits (mapped to EF)
# 1a) ig.EF (an object of class "igraph" storing as a directed graph)
g <- xRDataLoader('ig.EF')
g

# 1b) load GWAS SNPs annotated by EF (an object of class "dgCMatrix" storing a spare matrix)
anno <- xRDataLoader(RData='GWAS2EF')

# 1c) prepare the input SNPs of interest (eg 8 randomly chosen SNPs)
allSNPs <- rownames(anno)
data <- sample(allSNPs,8)

# 1d) perform calculate pair-wise semantic similarity between 8 randomly chosen SNPs
sim <- xSocialiser(data=data, annotation=anno, g=g, parallel=FALSE,
verbose=TRUE)
sim

# 1e) save similarity results to the file called 'EF_similarity.txt'
output <- igraph::get.data.frame(sim, what="edges")
utils::write.table(output, file="EF_similarity.txt", sep="\t",
row.names=FALSE)

# 1f) visualise the SNP network
## extract edge weight (with 2-digit precision)
x <- signif(as.numeric(E(sim)$weight), digits=2)
## rescale into an interval [1,4] as edge width
edge.width <- 1 + (x-min(x))/(max(x)-min(x))*3
## do visualisation
xVisNet(g=sim, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

## End(Not run)
```

---

xSocialiserGenes *Function to calculate pair-wise semantic similarity given a list of genes and the ontology in query*

Description

xSocialiserGenes is supposed to calculate pair-wise semantic similarity between a list of input SNPs and the ontology in query. It returns an object of class "igraph", a network representation of socialized genes. Now it supports enrichment analysis using a wide variety of ontologies such as Gene Ontology and Phenotype Ontologies. It first calculates semantic similarity between terms and then derives semantic similarity from term-term semantic similarity. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
xSocialiserGenes(data, ontology = c("GOBP", "GOMF", "GOCC", "DO",
"HPPA",
"HPMI", "HPCM", "HPMA", "MP"), measure = c("BM.average", "BM.max",
"BM.complete", "average", "max"), method.term = c("Resnik", "Lin",
"Schlicker", "Jiang", "Pesquita"), rescale = TRUE, force = TRUE,
fast = TRUE, parallel = TRUE, multicores = NULL,
path.mode = c("all_paths", "shortest_paths", "all_shortest_paths"),
true.path.rule = T, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

data	an input vector. It contains a list of Gene Symbols of interest
ontology	the ontology supported currently. It can be "GOBP" for Gene Ontology Biological Process, "GOMF" for Gene Ontology Molecular Function, "GOCC" for Gene Ontology Cellular Component, "DO" for Disease Ontology, "HPPA" for Human Phenotype Phenotypic Abnormality, "HPMI" for Human Phenotype Mode of Inheritance, "HPCM" for Human Phenotype Clinical Modifier, "HPMA" for Human Phenotype Mortality Aging, "MP" for Mammalian Phenotype
measure	the measure used to derive semantic similarity between genes/SNPs from semantic similarity between terms. Take the semantic similartity between SNPs as an example. It can be "average" for average similarity between any two terms (one from SNP 1, the other from SNP 2), "max" for the maximum similarity between any two terms, "BM.average" for best-matching (BM) based average similarity (i.e. for each term of either SNP, first calculate maximum similarity to any term in the other SNP, then take average of maximum similarity; the final BM-based average similiary is the pre-calculated average between two SNPs in pair), "BM.max" for BM based maximum similarity (i.e. the same as "BM.average", but the final BM-based maximum similiary is the maximum of the pre-calculated average between two SNPs in pair), "BM.complete" for BM-based complete-linkage similarity (inspired by complete-linkage concept: the least of any maximum similarity between a term of one SNP and a term of the other SNP). When comparing BM-based similarity between SNPs, "BM.average" and "BM.max" are sensitive to the number of terms invovled; instead, "BM.complete" is much robust in this aspect. By default, it uses "BM.average"
method.term	the method used to measure semantic similarity between terms. It can be "Resnik" for information content (IC) of most informative common ancestor (MICA) (see http://arxiv.org/pdf/cmp-lg/9511007.pdf), "Lin" for 2*IC at MICA divided by the sum of IC at pairs of terms (see https://www.cse.iitb.ac.in/~cs626-449/Papers/WordSimilarity/3.pdf), "Schlicker" for weighted version of 'Lin' by the 1-prob(MICA) (see http://www.ncbi.nlm.nih.gov/pubmed/16776819), "Jiang" for 1 - difference between the sum of IC at pairs of terms and 2*IC at MICA (see http://arxiv.org/pdf/cmp-lg/9709008.pdf), "Pesquita" for graph information content similarity related to Tanimoto-Jacard index (ie. summed information content of common ancestors divided by summed information content of all ancestors of term1 and term2 (see http://www.ncbi.nlm.nih.gov/pubmed/18460186))

rescale	logical to indicate whether the resulting values are rescaled to the range [0,1]. By default, it sets to true
force	logical to indicate whether the only most specific terms (for each SNP) will be used. By default, it sets to true. It is always advisable to use this since it is computationally fast but without compromising accuracy (considering the fact that true-path-rule has been applied when running xDAGanno)
fast	logical to indicate whether a vectorised fast computation is used. By default, it sets to true. It is always advisable to use this vectorised fast computation; since the conventional computation is just used for understanding scripts
parallel	logical to indicate whether parallel computation with multicores is used. By default, it sets to true, but not necessarily does so. Partly because parallel backends available will be system-specific (now only Linux or Mac OS). Also, it will depend on whether these two packages "foreach" and "doMC" have been installed. It can be installed via: `source("http://bioconductor.org/biocLite.R"); biocLite(c("foreach","doMC"))`. If not yet installed, this option will be disabled
multicores	an integer to specify how many cores will be registered as the multicore parallel backend to the 'foreach' package. If NULL, it will use a half of cores available in a user's computer. This option only works when parallel computation is enabled
path.mode	the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)
true.path.rule	logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display
RData.location	the characters to tell the location of built-in RData files. See xRDataLoader for details

Value

It returns an object of class "igraph", with nodes for input genes and edges for pair-wise semantic similarity between them.

Note

For the mode "shortest_paths", the induced subgraph is the most concise, and thus informative for visualisation when there are many nodes in query, while the mode "all_paths" results in the complete subgraph.

See Also

xSocialiser

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)
```

```
# Gene-based similarity analysis using Mammalian Phenotype Ontology (MP)
# a) provide the input Genes of interest (eg 100 randomly chosen human genes)
## load human genes
org.Hs.eg <- xRDataLoader(RData='org.Hs.eg')
data <- as.character(sample(org.Hs.eg$gene_info$Symbol, 100))
data

# b) perform similarity analysis
sim <- xSocialiserGenes(data=data, ontology="MP")

# c) save similarity results to the file called 'MP_similarity.txt'
output <- igraph::get.data.frame(sim, what="edges")
utils::write.table(output, file="MP_similarity.txt", sep="\t",
row.names=FALSE)

# d) visualise the gene network
## extract edge weight (with 2-digit precision)
x <- signif(as.numeric(E(sim)$weight), digits=2)
## rescale into an interval [1,4] as edge width
edge.width <- 1 + (x-min(x))/(max(x)-min(x))*3
## do visualisation
xVisNet(g=sim, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

## End(Not run)
```

`xSocialiserSNPs` *Function to calculate pair-wise semantic similarity given a list of SNPs and the ontology in query*

Description

`xSocialiserSNPs` is supposed to calculate pair-wise semantic similarity between a list of input SNPs and the ontology in query. It returns an object of class "igraph", a network representation of socialized SNPs. Now it supports analysis for SNPs using GWAS Catalog traits mapped to Experimental Factor Ontology. If required, additional SNPs that are in linkage disequilibrium (LD) with input SNPs are also be used for calculation. It first calculates semantic similarity between terms and then derives semantic similarity from term-term semantic similarity. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
xSocialiserSNPs(data, ontology = c("EF", "EF_disease", "EF_phenotype",
"EF_bp"), include.LD = NA, LD.r2 = 0.8, measure = c("BM.average",
"BM.max", "BM.complete", "average", "max"), method.term = c("Resnik",
"Lin",
"Schlicker", "Jiang", "Pesquita"), rescale = TRUE, force = TRUE,
fast = TRUE, parallel = TRUE, multicores = NULL,
path.mode = c("all_paths", "shortest_paths", "all_shortest_paths"),
true.path.rule = T, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

data	an input vector. It contains a list of SNPs of interest
ontology	the ontology supported currently. Now it is only "EF" for Experimental Factor Ontology (used to annotate GWAS Catalog SNPs). However, there are several subparts of this ontology to choose: 'EF_disease' for the subpart under the term 'disease' (EFO:0000408), 'EF_phenotype' for the subpart under the term 'phenotype' (EFO:0000651), 'EF_bp' for the subpart under the term 'biological process' (GO:0008150)
include.LD	additional SNPs in LD with input SNPs are also included. By default, it is 'NA' to disable this option. Otherwise, LD SNPs will be included based on one or more of 26 populations and 5 super populations from 1000 Genomics Project data (phase 3). The population can be one of 5 super populations ("AFR", "AMR", "EAS", "EUR", "SAS"), or one of 26 populations ("ACB", "ASW", "BEB", "CDX", "CEU", "CHB", "CHS", "CLM", "ESN", "FIN", "GBR", "GIH", "GWD", "IBS", "ITU", "JPT", "KHV", "LWK", "MSL", "MXL", "PEL", "PJL", "PUR", "STU", "TSI", "YRI"). Explanations for population code can be found at http://www.1000genomes.org/faq/which-populations-are-part-your-study
LD.r2	the LD r2 value. By default, it is 0.8, meaning that SNPs in LD (r2>=0.8) with input SNPs will be considered as LD SNPs. It can be any value from 0.8 to 1
measure	the measure used to derive semantic similarity between genes/SNPs from semantic similarity between terms. Take the semantic similartity between SNPs as an example. It can be "average" for average similarity between any two terms (one from SNP 1, the other from SNP 2), "max" for the maximum similarity between any two terms, "BM.average" for best-matching (BM) based average similarity (i.e. for each term of either SNP, first calculate maximum similarity to any term in the other SNP, then take average of maximum similarity; the final BM-based average similiary is the pre-calculated average between two SNPs in pair), "BM.max" for BM based maximum similarity (i.e. the same as "BM.average", but the final BM-based maximum similiary is the maximum of the pre-calculated average between two SNPs in pair), "BM.complete" for BM-based complete-linkage similarity (inspired by complete-linkage concept: the least of any maximum similarity between a term of one SNP and a term of the other SNP). When comparing BM-based similarity between SNPs, "BM.average" and "BM.max" are sensitive to the number of terms invovled; instead, "BM.complete" is much robust in this aspect. By default, it uses "BM.average"
method.term	the method used to measure semantic similarity between terms. It can be "Resnik" for information content (IC) of most informative common ancestor (MICA) (see http://arxiv.org/pdf/cmp-lg/9511007.pdf), "Lin" for 2*IC at MICA divided by the sum of IC at pairs of terms (see https://www.cse.iitb.ac.in/~cs626-449/Papers/WordSimilarity/3.pdf), "Schlicker" for weighted version of 'Lin' by the 1-prob(MICA) (see http://www.ncbi.nlm.nih.gov/pubmed/16776819), "Jiang" for 1 - difference between the sum of IC at pairs of terms and 2*IC at MICA (see http://arxiv.org/pdf/cmp-lg/9709008.pdf), "Pesquita" for graph information content similarity related to Tanimoto-Jacard index (ie. summed information content of common ancestors divided by summed information content of all ancestors of term1 and term2 (see http://www.ncbi.nlm.nih.gov/pubmed/18460186))
rescale	logical to indicate whether the resulting values are rescaled to the range [0,1]. By default, it sets to true
force	logical to indicate whether the only most specific terms (for each SNP) will be used. By default, it sets to true. It is always advisable to use this since it is

	computationally fast but without compromising accuracy (considering the fact that true-path-rule has been applied when running xDAGanno)
fast	logical to indicate whether a vectorised fast computation is used. By default, it sets to true. It is always advisable to use this vectorised fast computation; since the conventional computation is just used for understanding scripts
parallel	logical to indicate whether parallel computation with multicores is used. By default, it sets to true, but not necessarily does so. Partly because parallel backends available will be system-specific (now only Linux or Mac OS). Also, it will depend on whether these two packages "foreach" and "doMC" have been installed. It can be installed via: `source("http://bioconductor.org/biocLite.R"); biocLite(c("foreach","doMC"))`. If not yet installed, this option will be disabled
multicores	an integer to specify how many cores will be registered as the multicore parallel backend to the 'foreach' package. If NULL, it will use a half of cores available in a user's computer. This option only works when parallel computation is enabled
path.mode	the mode of paths induced by vertices/nodes with input annotation data. It can be "all_paths" for all possible paths to the root, "shortest_paths" for only one path to the root (for each node in query), "all_shortest_paths" for all shortest paths to the root (i.e. for each node, find all shortest paths with the equal lengths)
true.path.rule	logical to indicate whether the true-path rule should be applied to propagate annotations. By default, it sets to true
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to false for no display
RData.location	the characters to tell the location of built-in RData files. See xRDataLoader for details

Value

It returns an object of class "igraph", with nodes for input SNPs and edges for pair-wise semantic similarity between them.

Note

For the mode "shortest_paths", the induced subgraph is the most concise, and thus informative for visualisation when there are many nodes in query, while the mode "all_paths" results in the complete subgraph.

See Also

xSocialiser

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)

# SNP-based similarity analysis using GWAS Catalog traits (mapped to EF)
# a) provide the input SNPs of interest (eg 8 randomly chosen SNPs)
anno <- xRDataLoader(RData='GWAS2EF')
allSNPs <- rownames(anno)
```

```
data <- sample(allSNPs,8)
data

# b) perform similarity analysis
sim <- xSocialiserSNPs(data=data)

# b') optionally, enrichment analysis for input SNPs plus additional SNPs that are in LD with input SNPs
## LD based on European population (EUR) with r2>=0.8
#sim <- xSocialiserSNPs(data=data, include.LD="EUR", LD.r2=0.8)

# c) save similarity results to the file called 'EF_similarity.txt'
output <- igraph::get.data.frame(sim, what="edges")
utils::write.table(output, file="EF_similarity.txt", sep="\t",
row.names=FALSE)

# d) visualise the SNP network
## extract edge weight (with 2-digit precision)
x <- signif(as.numeric(E(sim)$weight), digits=2)
## rescale into an interval [1,4] as edge width
edge.width <- 1 + (x-min(x))/(max(x)-min(x))*3
## do visualisation
xVisNet(g=sim, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

## End(Not run)
```

---

xSubneter *Function to identify a subnetwork from an input network and the significance level imposed on its nodes*

Description

xSubneter is supposed to identify maximum-scoring subnetwork from an input graph with the node information on the significance (measured as p-values or fdr). It returns an object of class "igraph".

Usage

```
xSubneter(data, network = c("STRING", "PCOMMONS_UN", "PCOMMONS_DN"),
network.confidence = c("highest", "high", "medium"),
subnet.significance = 0.01, subnet.size = NULL, verbose = T,
RData.location =
"https://github.com/hfang-bristol/RDataCentre/blob/master/XGR/0.99.0")
```

Arguments

data	a named input vector containing the sinificance level for nodes (gene symbols). For this named vector, the element names are gene symbols, the element values for the significance level (measured as p-value or fdr). Alternatively, it can be a matrix or data frame with two columns: 1st column for gene symbols, 2nd column for the significance level
network	the ontology supported currently. It can be "STRING" for human functional protein association network from the STRING database (version 10), "PCOMMONS_UN" for human protein interaction undirected network from Pathway

Commons (version 7), "PCOMMONS_DN" for human protein interaction directed network from Pathway Commons (version 7)

`network.confidence`
: the parameter used to control the level of confidence of network edges that are to be further used. When "network" is "STRING", network edges with highest confidence are those with confidence scores=900, high confidence for confidence scores=700, medium confidence for confidence scores=400. When "network" is either "PCOMMONS_UN" or "PCOMMONS_DN", network edges with highest confidence are those supported with the PubMed references plus at least 3 different data sources, high confidence for those supported with the PubMed references plus at least 2 different data sources, medium confidence for those supported with at least 2 different data sources

`subnet.significance`
: the given significance threshold. By default, it is set to NULL, meaning there is no constraint on nodes/genes. If given, those nodes/genes with p-values below this are considered significant and thus scored positively. Instead, those p-values above this given significance threshold are considered insigificant and thus scored negatively

`subnet.size`
: the desired number of nodes constrained to the resulting subnet. It is not nulll, a wide range of significance thresholds will be scanned to find the optimal significance threshold leading to the desired number of nodes in the resulting subnet. Notably, the given significance threshold will be overwritten by this option

`verbose`
: logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

`RData.location`
: the characters to tell the location of built-in RData files. See `xRDataLoader` for details

Value

a subgraph with a maximum score, an object of class "igraph"

Note

The algorithm identifying a subnetwork is implemented in the dnet package (http://genomemedicine.biomedcentral.com/a
014-0064-8). In brief, from an input network with input node/gene information (the significant level; p-values or FDR), the way of searching for a maximum-scoring subnetwork is done as follows. Given the threshold of tolerable p-value, it gives positive scores for nodes with p-values below the threshold (nodes of interest), and negative scores for nodes with threshold-above p-values (intolerable). After score transformation, the search for a maximum scoring subnetwork is deduced to find the connected subnetwork that is enriched with positive-score nodes, allowing for a few negative-score nodes as linkers. This objective is met through minimum spanning tree finding and post-processing, previously used as a heuristic solver of prize-collecting Steiner tree problem. The solver is deterministic, only determined by the given tolerable p-value threshold. For identification of the subnetwork with a desired number of nodes, an iterative procedure is also developed to fine-tune tolerable thresholds. This explicit control over the node size may be necessary for guiding follow-up experiments.

See Also

`xRDataLoader`

Examples

```
## Not run:
# Load the library
library(XGR)
library(igraph)
library(dnet)

# a) provide the input nodes/genes with the significance info
## load human genes
org.Hs.eg <- xRDataLoader(RData='org.Hs.eg')
sig <- rbeta(500, shape1=0.5, shape2=1)
data <- data.frame(symbols=org.Hs.eg$gene_info$Symbol[1:500], sig)

# b) perform network analysis
# b1) find maximum-scoring subnet based on the given significance threshold
subnet <- xSubneter(data=data, network="STRING",
subnet.significance=0.01)
# b2) find maximum-scoring subnet with the desired node number=50
subnet <- xSubneter(data=data, network="STRING", subnet.size=50)

# c) save subnet results to the files called 'subnet_edges.txt' and 'subnet_nodes.txt'
output <- igraph::get.data.frame(subnet, what="edges")
utils::write.table(output, file="subnet_edges.txt", sep="\t",
row.names=FALSE)
output <- igraph::get.data.frame(subnet, what="vertices")
utils::write.table(output, file="subnet_nodes.txt", sep="\t",
row.names=FALSE)

# d) visualise the identified subnet
## do visualisation
xVisNet(g=subnet, pattern=V(subnet)$score, vertex.shape="sphere")

# e) visualise the identified subnet as a circos plot
library(RCircos)
library(GenomicRanges)
xCircos(g=subnet, entity="Gene")

## End(Not run)
```

`xVisNet` *Function to visualise a graph object of class "igraph"*

Description

`xVisNet` is supposed to visualise a graph object of class "igraph". It also allows vertices/nodes color-coded according to the input pattern.

Usage

```
xVisNet(g, pattern = NULL, colormap = c("bwr", "jet", "gbr", "wyr",
"br",
"yr", "rainbow", "wb"), ncolors = 40, zlim = NULL, colorbar = T,
newpage = T, glayout = layout.fruchterman.reingold,
```

```
vertex.frame.color = NA, vertex.size = NULL, vertex.color = NULL,
vertex.shape = NULL, vertex.label = NULL, vertex.label.cex = NULL,
vertex.label.dist = NULL, vertex.label.color = "black",
edge.arrow.size = 1, ...)
```

Arguments

g	an object of class "igraph"
pattern	a numeric vector used to color-code vertices/nodes. Notably, if the input vector contains names, then these names should include all node names of input graph, i.e. V(g)$name, since there is a mapping operation. After mapping, the length of the patern vector should be the same as the number of nodes of input graph; otherwise, this input pattern will be ignored. The way of how to color-code is to map values in the pattern onto the whole colormap (see the next arguments: colormap, ncolors, zlim and colorbar)
colormap	short name for the colormap. It can be one of "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-red colormap), "yr" (yellow-red colormap), "wb" (white-black colormap), and "rainbow" (rainbow colormap, that is, red-yellow-green-cyan-blue-magenta). Alternatively, any hyphen-separated HTML color names, e.g. "blue-black-yellow", "royalblue-white-sandybrown", "darkgreen-white-darkviolet". A list of standard color names can be found in http://html-color-codes.info/color-names
ncolors	the number of colors specified over the colormap
zlim	the minimum and maximum z/patttern values for which colors should be plotted, defaulting to the range of the finite values of z. Each of the given colors will be used to color an equispaced interval of this range. The midpoints of the intervals cover the range, so that values just outside the range will be plotted
colorbar	logical to indicate whether to append a colorbar. If pattern is null, it always sets to false
newpage	logical to indicate whether to open a new page. By default, it sets to true for opening a new page
glayout	either a function or a numeric matrix configuring how the vertices will be placed on the plot. If layout is a function, this function will be called with the graph as the single parameter to determine the actual coordinates. This function can be one of "layout.auto", "layout.random", "layout.circle", "layout.sphere", "layout.fruchterman.reingold", "layout.kamada.kawai", "layout.spring", "layout.reingold.tilford", "layout.fruchterman.reingold.grid", "layout.lgl", "layout.graphopt", "layout.svd" and "layout.norm". A full explanation of these layouts can be found in http://igraph.org/r/doc/layout_nicely.html
vertex.frame.color	the color of the frame of the vertices. If it is NA, then there is no frame
vertex.size	the size of each vertex. If it is a vector, each vertex may differ in size
vertex.color	the fill color of the vertices. If it is NA, then there is no fill color. If the pattern is given, this setup will be ignored
vertex.shape	the shape of each vertex. It can be one of "circle", "square", "csquare", "rectangle", "crectangle", "vrectangle", "pie" (http://igraph.org/r/doc/vertex.shape.pie.html), "sphere", and "none". If it sets to NULL, these vertices with negative will be "csquare" and the rest "circle".

`vertex.label` the label of the vertices. If it is NA, then there is no label. The default vertex labels are the name attribute of the nodes

`vertex.label.cex` the font size of vertex labels.

`vertex.label.dist` the distance of the label from the center of the vertex. If it is 0 then the label is centered on the vertex. If it is 1 then the label is displayed beside the vertex.

`vertex.label.color` the color of vertex labels.

`edge.arrow.size` the size of the arrows for the directed edge. The default value is 1.

`...` additional graphic parameters. See http://igraph.org/r/doc/plot.common.html for the complete list.

Value

invisible

Note

none

See Also

xSubneter

Examples

```
# 1) generate a ring graph
g <-make_ring(10, directed=TRUE)

# 2) visualise the graph
xVisNet(g=g, vertex.shape="sphere")

# 3) visualise the graph with vertices being color-coded by the pattern
pattern <- runif(vcount(g))
names(pattern) <- V(g)$name
xVisNet(g=g, pattern=pattern, colormap="bwr", vertex.shape="sphere")
```

Index