Package ‘dcGOR’

November 27, 2014

Type Package

Title Analysis of ontologies and protein domain annotations.

Version 1.0.4

Date 2014-11-27

Author Hai Fang and Julian Gough

Maintainer Hai Fang <hfang@cs.bris.ac.uk>

Depends R (>= 3.1.0), Matrix, igraph, dnet

Imports methods

Suggests foreach, doMC, ape

Description The package is used to analyse domain-centric ontologies and annotations, particularly those in the dcGO database. The dcGO (http://supfam.org/SUPERFAMILY/dcGO) is a comprehensive domain-centric database for annotating protein domains using a panel of ontologies including Gene Ontology. With the package, users are expected to analyse and visualise domain-centric ontologies and annotations. Supported analyses include but are not limited to: easy access to a wide range of ontologies and their domain-centric annotations; able to build customised ontologies and annotations; domain-based enrichment analysis and visualisation; construction of a domain (semantic similarity) network according to ontology annotations; significance analysis for estimating a contact (statistical significance) network via Random Walk with Restart; and high-performance parallel computing. The under-development functionality is: 1) to predict ontology terms for input protein sequences, precisely domain content in the form of architectures; 2) to reconstruct ancestral discrete characters using maximum likelihood/parsimony.

URL http://supfam.org/dcGOR, http://dcgor.r-forge.r-project.org, http://cran.r-project.org/package=dcGOR, https://github.com/hfang-bristol/dcGOR

Collate 'ClassMethod-dcGOR.r' 'dcDAGannotate.r' 'dcRDataLoader.r' 'dcEnrichment.r' 'visEnrichment.r' 'dcDAGdomainSim.r' 'dcRWRpipeline.r' 'dcConverter.r' 'dcBuildInfoDataFrame.r' 'dcBuildAnno.r' 'dcBuildOnto.r' 'dcAlgoPropagate.r' 'dcAlgoPredict.r' 'dcAlgoPredictMain.r' 'dcAlgoPredictGenome.r' 'dcAncestralML.r' 'dcAncestralMP.r' 'dcSubtreeClade.r' 'dcSubtreeTips.r' 'dcTreeConnectivity.r' 'dcDuplicated.r' 'dcSplitArch.r' 'dcAlgo.r'

License GPL-2

biocViews Bioinformatics

R topics documented:

AdjData-class *Definition for VIRTUAL S4 class AdjData*

Description

AdjData is union of other classes: either matrix or dgCMatrix (a sparse matrix in the package Matrix). It is used as a virtual class

Value

Class AdjData

See Also

Onto-class

Ancestral_domainome *Ancestral superfamily domain repertoires in Eukaryotes*

Description

An object of class "Anno" that contains information about domain repertoires (a complete set of domains: domain-ome) in Eukaryotes (including extant and ancestral genomes). This data is prepared based on 1) SUPERFAMILY database which provides domain and architecture assignments to all completely sequenced genomes including eukaryotic genomes; 2) ancestral domain architecture repertoires inferred by applying Dollo parsimony to eukaryotic part of species tree of life (sTOL), from which ancestral superfamily domain and architecture repertoires at all branching points in eukaryotic evolution are inferred. This allows us to list ancestral domain and architecture repertoires that were present at these points. Based on the observed/inferred domain and architecture repertoires, we also define genome-specific plasticity potential for an individual domain as how many different architectures (or architecture diversity) it can be formed in an extant/ancestral genome. As a result, for each genome, domain repertoires (domainome) are represented as a profile of states on domains, where non-zero entry indicates a domain for which how many different architectures have occurred in the genome.

Usage

```
data(Ancestral_domainome)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of 2019 domains X 875 terms/genomes (including 438 extant genomes and 437 ancestral genomes), with each entry telling how many different architectures a domain has in a genome. Note: zero entry also means that this domain is absent in the genome

- `termData`: variables describing terms/genomes (i.e. columns in annoData), including extant/ancestral genome information: "left_id" (unique and used as internal id), "right_id" (used in combination with "left_id" to define the post-ordered binary tree structure), "taxon_id" (NCBI taxonomy id, if matched), "genome" (2-letter genome identifiers used in SUPERFAMILY, if being extant), "name" (NCBI taxonomy name, if matched), "rank" (NCBI taxonomy rank, if matched), "branchlength" (branch length in relevance to the parent), and "common_name" (NCBI taxonomy common name, if matched and existed)
- `domainData`: variables describing domains (i.e. rows in annoData), including information about domains: "sunid" for SCOP id, "level" for SCOP level, "classification" for SCOP classification, "description" for SCOP description

References

Fang et al. (2013) A daily-updated tree of (sequenced) life as a reference for genome research. *Scientific reports*, 3:2015.
Morais et al. (2011) SUPERFAMILY 1.75 including a domain-centric gene ontology method. *Nucleic Acids Res*, 39(Database issue):D427-34.
Andreeva et al. (2008) Data growth and its impact on the SCOP database: new developments. *Nucleic Acids Res*, 36(Database issue):D419-425

Examples

```
data(Ancestral_domainome)
Ancestral_domainome
# retrieve info on terms/genomes
termData(Ancestral_domainome)
# retrieve info on SCOP domains
domainData(Ancestral_domainome)
# retrieve the first 5 rows and columns of data
x <- annoData(Ancestral_domainome)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

`Anno-class` *Definition for S4 class Anno*

Description

`Anno` has 3 slots: annoData, termData and domainData

Value

Class Anno

Slots

`annoData` An object of S4 class `AnnoData`, containing data matrix with the column number equal to nrow(termData) and the row number equal to nrow(domainData).

`termData` An object of S4 class `InfoDataFrame`, describing information on columns in annoData.

`domainData` An object of S4 class `InfoDataFrame`, describing information on rows in annoData.

Creation

An object of this class can be created via: `new("Anno", annoData,termData, domainData)`

Methods

Class-specific methods:

- `dim()`: retrieve the dimension in the object
- `annoData()`: retrieve the slot 'annoData' in the object
- `termData()`: retrieve the slot 'termData' (as class InfoDataFrame) in the object
- `domainData()`: retrieve the slot 'domainData' (as class InfoDataFrame) in the object
- `tData()`: retrieve termData (as data.frame) in the object
- `dData()`: retrieve domainData (as data.frame) in the object
- `termNames()`: retrieve term names (ie, row names of termData) in the object
- `domanNames()`: retrieve domain names (ie, row names of domainData) in the object

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object
- `as(matrix, "Anno")`: convert a matrix to an object of class Anno
- `as(dgCMatrix, "Anno")`: convert a dgCMatrix (a sparse matrix) to an object of class Anno
- `[i,j]`: get the subset of the same class

Access

Ways to access information on this class:

- `showClass("Anno")`: show the class definition
- `showMethods(classes="Anno")`: show the method definition upon this class
- `getSlots("Anno")`: get the name and class of each slot in this class
- `slotNames("Anno")`: get the name of each slot in this class
- `selectMethod(f, signature="Anno")`: retrieve the definition code for the method 'f' defined in this class

See Also

Anno-method

Examples

```
# create an object of class Anno, only given a matrix
annoData <- matrix(runif(50),nrow=10,ncol=5)
as(annoData, "Anno")

# create an object of class Anno, given a matrix plus information on its columns/rows
# 1) create termData: an object of class InfoDataFrame
data <- data.frame(x=1:5, y=I(LETTERS[1:5]), row.names=paste("Term",
1:5, sep="_"))
termData <- new("InfoDataFrame", data=data)
```

```
termData
# 2) create domainData: an object of class InfoDataFrame
data <- data.frame(x=1:10, y=I(LETTERS[1:10]),
row.names=paste("Domain", 1:10, sep="_"))
domainData <- new("InfoDataFrame", data=data)
domainData
# 3) create an object of class Anno
# VERY IMPORTANT: make sure having consistent names between annoData and domainData (and termData)
annoData <- matrix(runif(50),nrow=10,ncol=5)
rownames(annoData) <- rowNames(domainData)
colnames(annoData) <- rowNames(termData)
x <- new("Anno", annoData=annoData, domainData=domainData,
termData=termData)
x
# 4) look at various methods defined on class Anno
dim(x)
annoData(x)
termData(x)
tData(x)
domainData(x)
dData(x)
termNames(x)
domainNames(x)
# 5) get the subset
x[1:3,1:2]
```

Anno-method   *Methods defined for S4 class Anno*

Description

Methods defined for class Anno.

Usage

```
## S4 method for signature Anno
dim(x)

## S4 method for signature Anno
annoData(x)

## S4 method for signature Anno
termData(x)

## S4 method for signature Anno
domainData(x)

## S4 method for signature Anno
tData(object)

## S4 method for signature Anno
dData(object)
```

```
## S4 method for signature Anno
termNames(object)

## S4 method for signature Anno
domainNames(object)

## S4 method for signature Anno
show(object)

## S4 method for signature Anno,ANY,ANY,ANY
x[i, j, ..., drop = FALSE]
```

Arguments

x	an object of class Anno
object	an object of class Anno
i	an index
j	an index
...	additional parameters
drop	a logic for matrices and arrays. If TRUE the result is coerced to the lowest possible dimension. This only works for extracting elements, not for the replacement

See Also

Anno-class

AnnoData-class *Definition for VIRTUAL S4 class AnnoData*

Description

AnnoData is union of other classes: either matrix or dgCMatrix (a sparse matrix in the package Matrix). It is used as a virtual class

Value

Class AnnoData

See Also

Anno-class

`Cnetwork-class` *Definition for S4 class Cnetwork*

Description

`Cnetwork` is an S4 class to store a contact network, such as the one from RWR-based contact between samples/terms by `dcRWRpipeline`. It has 2 slots: nodeInfo and adjMatrix

Value

Class Cnetwork

Slots

`nodeInfo` An object of S4 class `InfoDataFrame`, describing information on nodes/domains.

`adjMatrix` An object of S4 class `AdjData`, containing symmetric adjacency data matrix for an indirect domain network

Creation

An object of this class can be created via: `new("Cnetwork",nodeInfo, adjMatrix)`

Methods

Class-specific methods:

- `dim()`: retrieve the dimension in the object
- `adjMatrix()`: retrieve the slot 'adjMatrix' in the object
- `nodeInfo()`: retrieve the slot 'nodeInfo' (as class InfoDataFrame) in the object
- `nInfo()`: retrieve nodeInfo (as data.frame) in the object
- `nodeNames()`: retrieve node/term names (ie, row names of nodeInfo) in the object

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object
- `as(matrix, "Cnetwork")`: convert a matrix to an object of class Cnetwork
- `as(dgCMatrix, "Cnetwork")`: convert a dgCMatrix (a sparse matrix) to an object of class Cnetwork
- `[i]`: get the subset of the same class

Access

Ways to access information on this class:

- `showClass("Cnetwork")`: show the class definition
- `showMethods(classes="Cnetwork")`: show the method definition upon this class
- `getSlots("Cnetwork")`: get the name and class of each slot in this class
- `slotNames("Cnetwork")`: get the name of each slot in this class
- `selectMethod(f, signature="Cnetwork")`: retrieve the definition code for the method 'f' defined in this class

See Also

Cnetwork-method

Examples

```
# create an object of class Cnetwork, only given a matrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
as(adjM, "Cnetwork")

# create an object of class Cnetwork, given a matrix plus information on nodes
# 1) create nodeI: an object of class InfoDataFrame
data <- data.frame(id=paste("Domain", 1:5, sep="_"),
level=rep("SCOP",5), description=I(LETTERS[1:5]),
row.names=paste("Domain", 1:5, sep="_"))
nodeI <- new("InfoDataFrame", data=data)
nodeI
# 2) create an object of class Cnetwork
# VERY IMPORTANT: make sure having consistent names between nodeInfo and adjMatrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
colnames(adjM) <- rownames(adjM) <- rowNames(nodeI)
x <- new("Cnetwork", adjMatrix=adjM, nodeInfo=nodeI)
x
# 3) look at various methods defined on class Cnetwork
dim(x)
adjMatrix(x)
nodeInfo(x)
nInfo(x)
nodeNames(x)
# 4) get the subset
x[1:2]
```

Cnetwork-method *Methods defined for S4 class Cnetwork*

Description

Methods defined for class Cnetwork.

Usage

```
## S4 method for signature Cnetwork
dim(x)

## S4 method for signature Cnetwork
adjMatrix(x)

## S4 method for signature Cnetwork
nodeInfo(x)

## S4 method for signature Cnetwork
nInfo(object)

## S4 method for signature Cnetwork
```

```
nodeNames(object)

## S4 method for signature Cnetwork
show(object)

## S4 method for signature Cnetwork,ANY,ANY,ANY
x[i, j, ..., drop = FALSE]
```

Arguments

x	an object of class Cnetwork
object	an object of class Cnetwork
i	an index
j	an index
...	additional parameters
drop	a logic for matrices and arrays. If TRUE the result is coerced to the lowest possible dimension. This only works for extracting elements, not for the replacement

See Also

Cnetwork-class

Coutput-class *Definition for S4 class Coutput*

Description

Coutput is an S4 class to store output by dcRWRpipeline.

Value

Class Coutput

Slots

ratio A symmetrix matrix, containing ratio

zscore A symmetrix matrix, containing z-scores

pvalue A symmetrix matrix, containing p-values

adjp A symmetrix matrix, containing adjusted p-values

cnetwork An object of S4 class Cnetwork, storing contact network.

Creation

An object of this class can be created via: new("Coutput", ratio,zscore, pvalue, adjp, cnetwork)

Methods

Class-specific methods:

- `ratio()`: retrieve the slot 'ratio' in the object
- `zscore()`: retrieve the slot 'zscore' in the object
- `pvalue()`: retrieve the slot 'pvalue' in the object
- `adjp()`: retrieve the slot 'adjp' in the object
- `cnetwork()`: retrieve the slot 'cnetwork' in the object
- `write()`: write the object into a local file

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object

Access

Ways to access information on this class:

- `showClass("Coutput")`: show the class definition
- `showMethods(classes="Coutput")`: show the method definition upon this class
- `getSlots("Coutput")`: get the name and class of each slot in this class
- `slotNames("Coutput")`: get the name of each slot in this class
- `selectMethod(f, signature="Coutput")`: retrieve the definition code for the method 'f' defined in this class

See Also

Coutput-method

Examples

```
# 1) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)

# 2) load SCOP superfamilies annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(SCOP.sf2GOMF)

# 3) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=TRUE)

# 4) calculate pair-wise semantic similarity between 10 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,10)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork

# 5) estimate RWR dating based sample/term relationships
# define sets of seeds as data
# each seed with equal weight (i.e. all non-zero entries are 1)
```

```
data <- data.frame(aSeeds=c(1,0,1,0,1), bSeeds=c(0,0,1,0,1))
rownames(data) <- id(dnetwork)[1:5]
# calcualte their two contact graph
coutput <- dcRWRpipeline(data=data, g=dnetwork, parallel=FALSE)
coutput

# 6) write into the file Coutput.txt in your local directory
write(coutput, file=Coutput.txt, saveBy="adjp")

# 7) retrieve several slots directly
ratio(coutput)
zscore(coutput)
pvalue(coutput)
adjp(coutput)
cnetwork(coutput)
```

`Coutput-method` *Methods defined for S4 class Coutput*

Description

Methods defined for S4 class `Coutput`.

Usage

```
## S4 method for signature Coutput
show(object)

## S4 method for signature Coutput
ratio(x)

## S4 method for signature Coutput
zscore(x)

## S4 method for signature Coutput
pvalue(x)

## S4 method for signature Coutput
adjp(x)

## S4 method for signature Coutput
cnetwork(x)

## S4 method for signature Coutput
write(x, file = "Coutput.txt", saveBy = c("adjp",
"pvalue", "zscore", "ratio"), verbose = T)
```

Arguments

`object`	an object of S4 class `Coutput`. Usually this is an output from `dcRWRpipeline`
`x`	an object of S4 class `Coutput`. Usually this is part of the output from `dcRWRpipeline`
`file`	a character specifying a file name written into. By default, it is 'Coutput.txt'

saveBy
: which statistics will be saved. It can be "pvalue" for p value, "adjp" for adjusted p value, "zscore" for z-score, "ratio" for ratio

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

Value

write(x) also returns a symmetrix matrix storing the specific statistics

See Also

Coutput-class

---

dcAlgo *Function to apply dcGO algorithm to infer domain-centric ontology*

Description

dcAlgo is supposed to apply dcGO algorithm to infer domain-centric ontology from input files. It requires two input files: 1) an annotation file containing annotations between proteins/genes and ontology terms; 2) an architecture file containing domain architectures for proteins/genes.

Usage

```
dcAlgo(anno.file, architecture.file, output.file = NULL, ontology =
c(NA,
"GOBP", "GOMF", "GOCC", "DO", "HPPA", "HPMI", "HPON", "MP", "EC", "KW",
"UP"),
feature.mode = c("supra", "individual", "comb"), min.overlap = 3,
fdr.cutoff = 0.001, parallel = TRUE, multicores = NULL, verbose = T,
RData.ontology.customised = NULL,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

anno.file
: an annotation file containing annotations between proteins/genes and ontology terms. For example, a file containing annotations between human genes and HP terms can be found in http://dcgor.r-forge.r-project.org/data/Algo/HP_anno.txt. As seen in this example, the input file must contain the header (in the first row) and two columns: 1st column for 'SeqID' (actually these IDs can be anything), 2nd column for 'termID' (HP terms)

architecture.file
: an architecture file containing domain architectures (including individual domains) for proteins/genes. For example, a file containing human genes and domain architectures can be found in http://dcgor.r-forge.r-project.org/data/Algo/SCOP_architecture.txt. As seen in this example, the input file must contain the header (in the first row) and two columns: 1st column for 'SeqID' (actually these IDs can be anything), 2nd column for 'Architecture' (SCOP domain architectures, each represented as comma-separated domains)

`output.file` an output file containing results. If not NULL, a tab-delimited text file will be also written out, with 1st column 'Feature_id' for features/domains, 2nd column 'Term_id' for ontology terms, 3rd column 'Score' for hypergeometric scores (indicative of strength for feature-term associations). Otherwise, there is no output file (by default)

`ontology` the ontology identity. 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, "HPON" for Human Phenotype ONset and clinical course, "MP" for Mammalian Phenotype, "EC" for Enzyme Commission, "KW" for UniProtKB KeyWords, "UP" for UniProtKB UniPathway. For details on the eligibility for pairs of input domain and ontology, please refer to the online Documentations at `http://supfam.org/dcGOR/docs.html`. If NA, then the user has to input a customised RData-formatted file (see `RData.ontology.customised` below)

`feature.mode` the mode of how to define the features thereof. It can be: "supra" for combinations of one or two successive domains (including individual domains; considering the order), "individual" for individual domains only, and "comb" for all possible combinations (including individual domains; ignoring the order)

`min.overlap` the minimum number of overlaps with each term in consideration. By default, it sets to a minimum of 3

`fdr.cutoff` the fdr cutoff to call the significant associations between features and terms. By default, it sets to 1e-3

`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

`RData.ontology.customised`
a file name for RData-formatted file containing an object of S4 class 'Onto' (i.g. ontology). By default, it is NULL. It is only needed when the user wants to perform customised analysis using their own ontology. See `dcBuildOnto` for how to creat this object

`RData.location` the characters to tell the location of built-in RData files. By default, it remotely locates at "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

Value

a data frame containing three columns: 1st column 'Feature_id' for features, 2nd 'Term_id' for terms, and 3rd 'Score' for the hypergeometric score indicative of strength of associations beteen features and terms

Note

When 'output.file' is specified, a tab-delimited text file is output, with the column names: 1st column 'Feature_id' for features, 2nd 'Term_id' for terms, and 3rd 'Score' for the hypergeometric score indicative of strength of associations beteen features and terms

See Also

dcRDataLoader, dcSplitArch, dcConverter, dcDuplicated, dcAlgoPropagate

Examples

```
# 1) Prepare input file: anno.file and architecture.file
anno.file <- "http://dcgor.r-forge.r-project.org/data/Algo/HP_anno.txt"
architecture.file <-
"http://dcgor.r-forge.r-project.org/data/Algo/SCOP_architecture.txt"

# 2) Do inference using built-in data
res <- dcAlgo(anno.file, architecture.file, ontology="HPPA",
feature.mode="supra", parallel=FALSE)
res[1:5,]

# 3) Advanced usage: using customised data
x <-
base::load(base::url("http://dcgor.r-forge.r-project.org/data/onto.HPPA.RData"))
RData.ontology.customised <- onto.HPPA.RData
base::save(list=x, file=RData.ontology.customised)
## you will see a RData file onto.HPPA.RData in local directory: list.files(pattern=*.RData)
res <- dcAlgo(anno.file, architecture.file, feature.mode="supra",
parallel=FALSE, RData.ontology.customised=RData.ontology.customised)
res[1:5,]
```

---

dcAlgoPredict *Function to predict ontology terms given domain architectures (including individual domains)*

Description

dcAlgoPredict is supposed to predict ontology terms given domain architectures (including individual domains). It involves 3 steps: 1) splitting an architecture into individual domains and all possible consecutive domain combinations (viewed as component features); 2) merging hscores among component features; 3) scaling merged hscores into predictive scores across terms.

Usage

```
dcAlgoPredict(data, RData.HIS = c(NA, "Feature2GOBP.sf",
"Feature2GOMF.sf",
"Feature2GOCC.sf", "Feature2HPPA.sf", "Feature2GOBP.pfam",
"Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam",
"Feature2GOBP.interpro", "Feature2GOMF.interpro",
"Feature2GOCC.interpro",
"Feature2HPPA.interpro"), merge.method = c("sum", "max", "sequential"),
scale.method = c("log", "linear", "none"), feature.mode = c("supra",
"individual", "comb"), slim.level = NULL, max.num = NULL,
parallel = TRUE, multicores = NULL, verbose = T,
RData.HIS.customised = NULL,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

`data` an input data vector containing domain architectures. An architecture is represented in the form of comma-separated domains

`RData.HIS` RData to load. This RData conveys two bits of information: 1) feature (domain) type; 2) ontology. It stores the hypergeometric scores (hscore) between features (individual domains or consecutive domain combinations) and ontology terms. The RData name tells which domain type and which ontology to use. It can be: SCOP sf domains/combinations (including "Feature2GOBP.sf", "Feature2GOMF.sf", "Feature2GOCC.sf", "Feature2HPPA.sf"), Pfam domains/combinations (including "Feature2GOBP.pfam", "Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam"), InterPro domains (including "Feature2GOBP.interpro", "Feature2GOMF.interpro", "Feature2GOCC.interpro", "Feature2HPPA.interpro"). If NA, then the user has to input a customised RData-formatted file (see `RData.HIS.customised` below)

`merge.method` the method used to merge predictions for each component feature (individual domains and their combinations derived from domain architecture). It can be one of "sum" for summing up, "max" for the maximum, and "sequential" for the sequential weighting. The sequential weighting is done via: $\sum_{i=1} \frac{R_i}{i}$, where R_i is the i^{th} ranked highest hscore

`scale.method` the method used to scale the predictive scores. It can be: "none" for no scaling, "linear" for being linearily scaled into the range between 0 and 1, "log" for the same as "linear" but being first log-transformed before being scaled. The scaling between 0 and 1 is done via: $\frac{S-S_{min}}{S_{max}-S_{min}}$, where S_{min} and S_{max} are the minimum and maximum values for S

`feature.mode` the mode of how to define the features thereof. It can be: "supra" for combinations of one or two successive domains (including individual domains; considering the order), "individual" for individual domains only, and "comb" for all possible combinations (including individual domains; ignoring the order)

`slim.level` whether only slim terms are returned. By defaut, it is NULL and all predicted terms will be reported. If it is specified as a vector containing any values from 1 to 4, then only slim terms at these levels will be reported. Here is the meaning of these values: '1' for very general terms, '2' for general terms, '3' for specific terms, and '4' for very specific terms

`max.num` whether only top terms per sequence are returned. By defaut, it is NULL and no constraint is imposed. If an integer is specified, then all predicted terms (with scores in a decreasing order) beyond this number will be discarded. Notably, this parameter works after the preceding parameter `slim.level`

`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

`RData.HIS.customised` a file name for RData-formatted file containing an object of S3 class 'HIS'. By default, it is NULL. It is only needed when the user wants to perform customised analysis. See `dcAlgoPropagate` on how this object is created

`RData.location` the characters to tell the location of built-in RData files. By default, it remotely locates at "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

Value

a named list of architectures, each containing predictive scores

Note

none

See Also

`dcRDataLoader`, `dcSplitArch`, `dcConverter`, `dcAlgoPropagate`, `dcAlgoPredictMain`, `dcAlgoPredictGenome`

Examples

```
# 1) randomly generate 5 domains and/or domain architectures
x <- dcRDataLoader(RData="Feature2GOMF.sf")
data <- sample(names(x$hscore), 5)

# 2) get predictive scores of all predicted terms for this domain architecture
## using sequential method (by default)
pscore <- dcAlgoPredict(data=data, RData.HIS="Feature2GOMF.sf",
parallel=FALSE)
## using max method
```

```
pscore_max <- dcAlgoPredict(data=data, RData.HIS="Feature2GOMF.sf",
merge.method="max", parallel=FALSE)
## using sum method
pscore_sum <- dcAlgoPredict(data=data, RData.HIS="Feature2GOMF.sf",
merge.method="sum", parallel=FALSE)

# 3) advanced usage
## a) focus on those terms at the 2nd level (general)
pscore <- dcAlgoPredict(data=data, RData.HIS="Feature2GOMF.sf",
slim.level=2, parallel=FALSE)
## b) visualise predictive scores in the ontology hierarchy
### load the ontology
g <- dcRDataLoader("onto.GOMF", verbose=FALSE)
ig <- dcConverter(g, from=Onto, to=igraph, verbose=FALSE)
### do visualisation for the 1st architecture
data <- pscore[[1]]
subg <- dnet::dDAGinduce(ig, nodes_query=names(data),
path.mode="shortest_paths")
dnet::visDAG(g=subg, data=data, node.info="term_id")
```

`dcAlgoPredictGenome` *Function to predict ontology terms for genomes with domain architectures (including individual domains)*

Description

`dcAlgoPredictGenome` is supposed to predict ontology terms for genomes with domain architectures (including individual domains).

Usage

```
dcAlgoPredictGenome(input.file, RData.HIS = c(NULL, "Feature2GOBP.sf",
"Feature2GOMF.sf", "Feature2GOCC.sf", "Feature2HPPA.sf",
"Feature2GOBP.pfam",
"Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam",
"Feature2GOBP.interpro", "Feature2GOMF.interpro",
"Feature2GOCC.interpro",
"Feature2HPPA.interpro"), weight.method = c("none", "copynum", "ic",
"both"), merge.method = c("sum", "max", "sequential"),
scale.method = c("log", "linear", "none"), feature.mode = c("supra",
"individual", "comb"), slim.level = NULL, max.num = NULL,
parallel = TRUE, multicores = NULL, verbose = T,
RData.HIS.customised = NULL,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

`input.file`	an input file containing genomes and their domain architectures (including individual domains). For example, a file containing Hominidae genomes and their domain architectures can be found in `http://dcgor.r-forge.r-project.org/data/Feature/Hominidae.txt`. As seen in this example, the input file must contain the header (in the first row) and two columns: 1st column for 'Genome'

(a genome like a container), 2nd column for 'Architecture' (SCOP domain architectures, each represented as comma-separated domains). Alternatively, the input.file can be a matrix or data frame, assuming that input file has been read

`RData.HIS` RData to load. This RData conveys two bits of information: 1) feature (domain) type; 2) ontology. It stores the hypergeometric scores (hscore) between features (individual domains or consecutive domain combinations) and ontology terms. The RData name tells which domain type and which ontology to use. It can be: SCOP sf domains/combinations (including "Feature2GOBP.sf", "Feature2GOMF.sf", "Feature2GOCC.sf", "Feature2HPPA.sf"), Pfam domains/combinations (including "Feature2GOBP.pfam", "Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam"), InterPro domains (including "Feature2GOBP.interpro", "Feature2GOMF.interpro", "Feature2GOCC.interpro", "Feature2HPPA.interpro"). If NA, then the user has to input a customised RData-formatted file (see `RData.HIS.customised` below)

`weight.method` the method used how to weight predictions. It can be one of "none" (no weighting; by default), "copynum" for weighting copynumber of architectures, and "ic" for weighting information content (ic) of the term, "both" for weighting both copynumber and ic

`merge.method` the method used to merge predictions for each component feature (individual domains and their combinations derived from domain architecture). It can be one of "sum" for summing up, "max" for the maximum, and "sequential" for the sequential merging. The sequential merging is done via: $\sum_{i=1} \frac{R_i}{i}$, where R_i is the i^{th} ranked highest hscore

`scale.method` the method used to scale the predictive scores. It can be: "none" for no scaling, "linear" for being linearily scaled into the range between 0 and 1, "log" for the same as "linear" but being first log-transformed before being scaled. The scaling between 0 and 1 is done via: $\frac{S-S_{min}}{S_{max}-S_{min}}$, where S_{min} and S_{max} are the minimum and maximum values for S

`feature.mode` the mode of how to define the features thereof. It can be: "supra" for combinations of one or two successive domains (including individual domains; considering the order), "individual" for individual domains only, and "comb" for all possible combinations (including individual domains; ignoring the order)

`slim.level` whether only slim terms are returned. By defaut, it is NULL and all predicted terms will be reported. If it is specified as a vector containing any values from 1 to 4, then only slim terms at these levels will be reported. Here is the meaning of these values: '1' for very general terms, '2' for general terms, '3' for specific terms, and '4' for very specific terms

`max.num` whether only top terms per sequence are returned. By defaut, it is NULL and no constraint is imposed. If an integer is specified, then all predicted terms (with scores in a decreasing order) beyond this number will be discarded. Notably, this parameter works after the preceding parameter `slim.level`

`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

RData.HIS.customised — a file name for RData-formatted file containing an object of S3 class 'HIS'. By default, it is NULL. It is only needed when the user wants to perform customised analysis. See dcAlgoPropagate on how this object is created

RData.location — the characters to tell the location of built-in RData files. By default, it remotely locates at "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function data first)

Value

a matrix of terms X genomes, containing the predicted scores (per genome) as a whole

Note

none

See Also

dcRDataLoader, dcAlgoPropagate, dcAlgoPredict

Examples

```
# 1) Prepare an input file containing domain architectures
input.file <-
"http://dcgor.r-forge.r-project.org/data/Feature/Hominidae.txt"

# 2) Do prediction using built-in data
output <- dcAlgoPredictGenome(input.file, RData.HIS="Feature2GOMF.sf",
parallel=FALSE)
dim(output)
output[1:10,]

# 3) Advanced usage: using customised data
x <-
base::load(base::url("http://dcgor.r-forge.r-project.org/data/Feature2GOMF.sf.RData"))
RData.HIS.customised <- Feature2GOMF.sf.RData
base::save(list=x, file=RData.HIS.customised)
## you will see a RData file Feature2GOMF.sf.RData in local directory: list.files(pattern=*.RData)
output <- dcAlgoPredictGenome(input.file, parallel=FALSE,
RData.HIS.customised=RData.HIS.customised)
dim(output)
output[1:10,]
```

dcAlgoPredictMain *Function to predict ontology terms given an input file containing domain architectures (including individual domains)*

Description

dcAlgoPredictMain is supposed to predict ontology terms given an input file containing domain architectures (including individual domains).

Usage

```
dcAlgoPredictMain(input.file, output.file = NULL, RData.HIS = c(NA,
"Feature2GOBP.sf", "Feature2GOMF.sf", "Feature2GOCC.sf",
"Feature2HPPA.sf",
"Feature2GOBP.pfam", "Feature2GOMF.pfam", "Feature2GOCC.pfam",
"Feature2HPPA.pfam", "Feature2GOBP.interpro", "Feature2GOMF.interpro",
"Feature2GOCC.interpro", "Feature2HPPA.interpro"), merge.method =
c("sum",
"max", "sequential"), scale.method = c("log", "linear", "none"),
feature.mode = c("supra", "individual", "comb"), slim.level = NULL,
max.num = NULL, parallel = TRUE, multicores = NULL, verbose = T,
RData.HIS.customised = NULL,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

input.file	an input file containing domain architectures (including individual domains). For example, a file containing UniProt ID and domain architectures for human proteins can be found in http://dcgor.r-forge.r-project.org/data/Feature/hs.txt. As seen in this example, the input file must contain the header (in the first row) and two columns: 1st column for 'SeqID' (actually these IDs can be anything), 2nd column for 'Architecture' (SCOP domain architectures, each represented as comma-separated domains). Alternatively, the input.file can be a matrix or data frame, assuming that input file has been read
output.file	an output file containing predicted results. If not NULL, a tab-delimited text file will be also written out; otherwise, there is no output file (by default)
RData.HIS	RData to load. This RData conveys two bits of information: 1) feature (domain) type; 2) ontology. It stores the hypergeometric scores (hscore) between features (individual domains or consecutive domain combinations) and ontology terms. The RData name tells which domain type and which ontology to use. It can be: SCOP sf domains/combinations (including "Feature2GOBP.sf", "Feature2GOMF.sf", "Feature2GOCC.sf", "Feature2HPPA.sf"), Pfam domains/combinations (including "Feature2GOBP.pfam", "Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam"), InterPro domains (including "Feature2GOBP.interpro", "Feature2GOMF.interpro", "Feature2GOCC.interpro", "Feature2HPPA.interpro"). If NA, then the user has to input a customised RData-formatted file (see RData.HIS.customised below)
merge.method	the method used to merge predictions for each component feature (individual domains and their combinations derived from domain architecture). It can be one of "sum" for summing up, "max" for the maximum, and "sequential" for the

sequential merging. The sequential merging is done via: $\sum_{i=1} \frac{R_i}{i}$, where R_i is the i^{th} ranked highest hscore

`scale.method` the method used to scale the predictive scores. It can be: "none" for no scaling, "linear" for being linearily scaled into the range between 0 and 1, "log" for the same as "linear" but being first log-transformed before being scaled. The scaling between 0 and 1 is done via: $\frac{S-S_{min}}{S_{max}-S_{min}}$, where S_{min} and S_{max} are the minimum and maximum values for S

`feature.mode` the mode of how to define the features thereof. It can be: "supra" for combinations of one or two successive domains (including individual domains; considering the order), "individual" for individual domains only, and "comb" for all possible combinations (including individual domains; ignoring the order)

`slim.level` whether only slim terms are returned. By defaut, it is NULL and all predicted terms will be reported. If it is specified as a vector containing any values from 1 to 4, then only slim terms at these levels will be reported. Here is the meaning of these values: '1' for very general terms, '2' for general terms, '3' for specific terms, and '4' for very specific terms

`max.num` whether only top terms per sequence are returned. By defaut, it is NULL and no constraint is imposed. If an integer is specified, then all predicted terms (with scores in a decreasing order) beyond this number will be discarded. Notably, this parameter works after the preceding parameter `slim.level`

`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

`RData.HIS.customised`
a file name for RData-formatted file containing an object of S3 class 'HIS'. By default, it is NULL. It is only needed when the user wants to perform customised analysis. See `dcAlgoPropagate` on how this object is created

`RData.location` the characters to tell the location of built-in RData files. By default, it remotely locates at "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

Value

a data frame containing three columns: 1st column the same as the input file (e.g. 'SeqID'), 2nd for 'Term' (predicted ontology terms), 3rd for 'Score' (along with predicted scores)

Note

When 'output.file' is specified, a tab-delimited text file is written out, with the column names: 1st column the same as the input file (e.g. 'SeqID'), 2nd for 'Term' (predicted ontology terms), 3rd for 'Score' (along with predicted scores)

See Also

dcRDataLoader, dcAlgoPropagate, dcAlgoPredict

Examples

```
# 1) Prepare an input file containing domain architectures
input.file <- "http://dcgor.r-forge.r-project.org/data/Feature/hs.txt"

# 2) Do prediction using built-in data
output <- dcAlgoPredictMain(input.file, RData.HIS="Feature2GOMF.sf",
parallel=FALSE)
output[1:5,]

# 3) Advanced usage: using customised data
x <-
base::load(base::url("http://dcgor.r-forge.r-project.org/data/Feature2GOMF.sf.RData"))
RData.HIS.customised <- Feature2GOMF.sf.RData
base::save(list=x, file=RData.HIS.customised)
## you will see a RData file Feature2GOMF.sf.RData in local directory: list.files(pattern=*.RData)
output <- dcAlgoPredictMain(input.file, parallel=FALSE,
RData.HIS.customised=RData.HIS.customised)
output[1:5,]
```

---

dcAlgoPropagate *Function to propogate ontology annotations according to an input file*

Description

dcAlgoPropagate is supposed to propogate ontology annotations, given an input file. This input file contains original annotations between domains/features and ontology terms, along with the hypergeometric scores (hscore) in support for their annotations. The annotations are propogated to the ontology root (retaining the maximum hscore). After the propogation, the ontology terms of increasing levels are determined based on the concept of Information Content (IC) to product a slim version of ontology. It returns an object of S3 class "HIS" with three components: "hscore", "ic" and "slim".

Usage

```
dcAlgoPropagate(input.file, ontology = c("GOBP", "GOMF", "GOCC", "DO",
"HPPA",
"HPMI", "HPON", "MP", "EC", "KW", "UP"), output.file = "HIS.RData",
verbose = T, RData.location =
"http://dcgor.r-forge.r-project.org/data")
```

Arguments

`input.file`	an input file used to build the object. This input file contains original annotations between domains/features and ontology terms, along with the hypergeometric scores (hscore) in support for their annotations. For example, a file containing original annotations between SCOP domain architectures and GO terms can be found in http://dcgor.r-forge.r-project.org/data/Feature/Feature2GO.sf.txt. As seen in this example, the input file must contain the header (in the first row) and three columns: 1st column for 'Feature_id' (here SCOP domain architectures), 2nd column for 'Term_id' (GO terms), and 3rd column for 'Score' (hscore)
`ontology`	the ontology identity. 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, "HPON" for Human Phenotype ONset and clinical course, "MP" for Mammalian Phenotype, "EC" for Enzyme Commission, "KW" for UniProtKB KeyWords, "UP" for UniProtKB UniPathway. For details on the eligibility for pairs of input domain and ontology, please refer to the online Documentations at http://supfam.org/dcGOR/docs.html
`output.file`	an output file used to save the `HIS` object as an RData-formatted file (see 'Value' for details). If NULL, this file will be saved into "HIS.RData" in the current working local directory
`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 "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

Value

an object of S3 class `HIS`, with following components:

- `hscore`: a list of features, each with a term-named vector containing hscore
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

Note

None

See Also

dcRDataLoader, dcConverter, dcAlgo

Examples

```
# build an "HIS" object for GO Molecular Function
Feature2GOMF.sf <-
dcAlgoPropagate(input.file="http://dcgor.r-forge.r-project.org/data/Feature/Feature2GO.sf.txt",
ontology="GOMF", output.file="Feature2GOMF.sf.RData")
names(Feature2GOMF.sf)
Feature2GOMF.sf$hscore[1]
Feature2GOMF.sf$ic[1:10]
Feature2GOMF.sf$slim[1]

# extract hscore as a matrix with 3 columns (Feature_id, Term_id, Score)
hscore <- Feature2GOMF.sf$hscore
output_list <- lapply(1:length(hscore), function(i){
x <- hscore[[i]]
y <- rep(names(hscore)[i],length(x))
cbind(Feature_id=y, Term_id=names(x), Score=as.numeric(x))
})
hscore_mat <- base::do.call(base::rbind, output_list)
dim(hscore_mat)
hscore_mat[1:10,]
```

---

dcAncestralML — *Function to reconstruct ancestral discrete states using fast maximum likelihood algorithm*

Description

dcAncestralML is supposed to reconstruct ancestral discrete states using fast maximum likelihood algorithm. It takes inputs both the phylo-formatted tree and discrete states in the tips. The algorithm assumes that state changes can be described by a probablistic reversible model. It first determines transition matrix between states (also considering branch lengths), then uses dynamic programming (from tips to the root) to estimate conditional maximum likelihood, and finally reconstructs the ancestral states (from the root to tips). If the ties occur at the root, the state at the root is set to the last state in ties (for example, usually being 'present' for 'present'-'absent' two states).

Usage

```
dcAncestralML(data, phy, transition.model = c("different", "symmetric",
"same", "customised"), customised.model = NULL, edge.length.power = 1,
initial.estimate = 0.1, output.detail = F, parallel = T,
multicores = NULL, verbose = T)
```

Arguments

data — an input data matrix storing discrete states for tips (in rows) X characters (in columns). The rows in the matrix are for tips. If the row names do not exist, then addumedly they have the same order as in the tree tips. More wisely, users provide row names which can be matched to the tip labels of the tree. The row names can be more than found in the tree labels, and they should contain all those in the tree labels

`phy` an object of class 'phylo'

`transition.model`
: a character specifying the transition model. It can be: "different" for all-transition-different model (such as $matrix(c(0,1,2,0),2)$), "symmetric" for the symmetric model (such as $matrix(c(0,1,1,0),2)$ or $matrix(c(0,1,2,1,0,3,2,3,0),3)$), "same" for all-transition-same model (such as $matrix(c(0,1,1,0),2)$), "customised" for the user-customised model (see the next parameter)

`customised.model`
: a matrix customised for the transition model. It can be: $matrix(c(0,1,1,0),2)$, $matrix(c(0,1,2,0),2)$, or $matrix(c(0,1,2,1,0,3,2,3,0),3)$

`edge.length.power`
: a non-negative value giving the exponent transformation of the branch lengths. It is useful when determining transition matrix between states

`initial.estimate`
: the initial value used for the maximum likelihood estimation

`output.detail` logical to indicate whether the output is returned as a detailed list. If TRUE, a nested list is returned: a list of characters (corresponding to columns of input data matrix), in which each element is a list consisting of three components ("states", "transition" and "relative"). If FALSE, a matrix is returned: the columns respond to the input data columns, and rows responding to all node index in the phylo-formatted tree

`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 depends on the 'output.detail'. If FALSE (by default), a matrix is returned, with the columns responding to the input data columns, and rows responding to node index in the phylo-formatted tree. If TRUE, a nested list is returned. Outer-most list is for characters (corresponding to columns of input data matrix), in which each elemenl is a list (inner-most) consisting of three components ("states", "transition" and "relative"):

- `states`: a named vector storing states (extant and ancestral states)
- `transition`: an estimated transition matrix between states
- `relative`: a matrix of nodes X states, storing conditional maximum likelihood being relative to each state

Note

This fast dynamic programming for ancestral discrete state reconstruction is partially inspired by a joint estimation procedure as described in http://mbe.oxfordjournals.org/content/17/6/890.full

See Also

dcAncestralMP, dcDuplicated

Examples

```
# 1) a newick tree that is imported as a phylo-formatted tree
tree <- "(((t1:5,t2:5):2,(t3:4,t4:4):3):2,(t5:4,t6:4):6);"
phy <- ape::read.tree(text=tree)

# 2) an input data matrix storing discrete states for tips (in rows) X four characters (in columns)
data1 <- matrix(c(0,rep(1,3),rep(0,2)), ncol=1)
data2 <- matrix(c(rep(0,4),rep(1,2)), ncol=1)
data <- cbind(data1, data1, data1, data2)
colnames(data) <- c("C1", "C2", "C3", "C4")
## reconstruct ancestral states, without detailed output
res <- dcAncestralML(data, phy, parallel=FALSE)
res

# 3) an input data matrix storing discrete states for tips (in rows) X only one character
data <- matrix(c(0,rep(1,3),rep(0,2)), ncol=1)
## reconstruct ancestral states, with detailed output
res <- dcAncestralML(data, phy, parallel=FALSE, output.detail=TRUE)
res
## get the inner-most list
res <- res[[1]]
## visualise the tree with ancestral states and their conditional probability
Ntip <- ape::Ntip(phy)
Nnode <- ape::Nnode(phy)
color <- c("white","gray")
## visualise main tree
ape::plot.phylo(phy, type="p", use.edge.length=TRUE, label.offset=1,
show.tip.label=TRUE, show.node.label=FALSE)
## visualise tips (state 1 in gray, state 0 in white)
x <- data[,1]
ape::tiplabels(pch=22, bg=color[as.numeric(x)+1], cex=2, adj=1)
## visualise internal nodes
### thermo bar to illustrate relative probability (state 1 in gray, state 0 in white)
ape::nodelabels(thermo=res$relative[Ntip+1:Nnode,2:1],
piecol=color[2:1], cex=0.75)
### labeling reconstructed ancestral states
ape::nodelabels(text=res$states[Ntip+1:Nnode], node=Ntip+1:Nnode,
frame="none", col="red", bg="transparent", cex=0.75)
```

---

dcAncestralMP	*Function to reconstruct ancestral discrete states using maximum parsimony algorithm*

Description

`dcAncestralMP` is supposed to reconstruct ancestral discrete states using a maximum parsimony-modified Fitch algorithm. In a from-tip-to-root manner, ancestral state for an internal node is determined if a state is shared in a majority by all its children. If two or more states in a majority are equally shared, this internal node is temporarily marked as an unknown tie, which is further resolved in a from-root-to-tip manner: always being the same state as its direct parent holds. If the ties also occur at the root, the state at the root is set to the last state in ties (for example, usually being 'present' for 'present'-'absent' two states).

Usage

```
dcAncestralMP(data, phy, output.detail = F, parallel = T,
multicores = NULL, verbose = T)
```

Arguments

`data`	an input data matrix/frame storing discrete states for tips (in rows) X characters (in columns). The rows in the matrix are for tips. If the row names do not exist, then addumedly they have the same order as in the tree tips. More wisely, users provide row names which can be matched to the tip labels of the tree. The row names can be more than found in the tree labels, and they should contain all those in the tree labels
`phy`	an object of class 'phylo'
`output.detail`	logical to indicate whether the output is returned as a detailed list. If TRUE, a nested list is returned: a list of characters (corresponding to columns of input data matrix), in which each element is a list consisting of three components ("states", "transition" and "relative"). If FALSE, a matrix is returned: the columns respond to the input data columns, and rows responding to all node index in the phylo-formatted tree
`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 depends on the 'output.detail'. If FALSE (by default), a matrix is returned, with the columns responding to the input data columns, and rows responding to node index in the phylo-formatted tree. If TRUE, a nested list is returned. Outer-most list is for characters (corresponding to columns of input data matrix), in which each elemenl is a list (inner-most) consisting of three components ("states", "transition" and "relative"):

- `states`: a named vector storing states (extant and ancestral states)

- `transition`: a posterior transition matrix between states
- `relative`: a matrix of nodes X states, storing relative probability

Note

This maximum parsimony algorithm for ancestral discrete state reconstruction is attributable to the basic idea as described in http://sysbio.oxfordjournals.org/content/20/4/406.short

See Also

`dcAncestralML`, `dcTreeConnectivity`, `dcDuplicated`

Examples

```
# 1) a newick tree that is imported as a phylo-formatted tree
tree <- "(((t1:5,t2:5):2,(t3:4,t4:4):3):2,(t5:4,t6:4):6);"
phy <- ape::read.tree(text=tree)

# 2) an input data matrix storing discrete states for tips (in rows) X four characters (in columns)
data1 <- matrix(c(0,rep(1,3),rep(0,2)), ncol=1)
data2 <- matrix(c(rep(0,4),rep(1,2)), ncol=1)
data <- cbind(data1, data1, data1, data2)
colnames(data) <- c("C1", "C2", "C3", "C4")
## reconstruct ancestral states, without detailed output
res <- dcAncestralMP(data, phy, parallel=FALSE)
res

# 3) an input data matrix storing discrete states for tips (in rows) X only one character
data <- matrix(c(0,rep(1,3),rep(0,2)), ncol=1)
## reconstruct ancestral states, with detailed output
res <- dcAncestralMP(data, phy, parallel=FALSE, output.detail=TRUE)
res
## get the inner-most list
res <- res[[1]]
## visualise the tree with ancestral states and their conditional probability
Ntip <- ape::Ntip(phy)
Nnode <- ape::Nnode(phy)
color <- c("white","gray")
## visualise main tree
ape::plot.phylo(phy, type="p", use.edge.length=TRUE, label.offset=1,
show.tip.label=TRUE, show.node.label=FALSE)
## visualise tips (state 1 in gray, state 0 in white)
x <- data[,1]
ape::tiplabels(pch=22, bg=color[as.numeric(x)+1], cex=2, adj=1)
## visualise internal nodes
### thermo bar to illustrate relative probability (state 1 in gray, state 0 in white)
ape::nodelabels(thermo=res$relative[Ntip+1:Nnode,2:1],
piecol=color[2:1], cex=0.75)
### labeling reconstructed ancestral states
ape::nodelabels(text=res$states[Ntip+1:Nnode], node=Ntip+1:Nnode,
frame="none", col="red", bg="transparent", cex=0.75)
```

dcBuildAnno *Function to build an object of the S4 class Anno from input files*

Description

dcBuildAnno is supposed to build an object of of the S4 class Anno, given input files. These input files include 1) a file containing domain information, 2) a file containing term information, and 3) a file containing associations between domains and terms.

Usage

```
dcBuildAnno(domain_info.file, term_info.file, association.file,
output.file = "Anno.RData")
```

Arguments

`domain_info.file`
: an input file containing domain information. For example, a file containing InterPro domains (InterPro) can be found in http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt. As seen in this example, the input file must contain the header (in the first row), and entries in the first column intend to be domain ID (and must be unique)

`term_info.file`
: an input file containing term information. For example, a file containing Gene Ontology (GO) terms can be found in http://dcgor.r-forge.r-project.org/data/InterPro/GO.txt. As seen in this example, the input file must contain the header (in the first row) and four columns: 1st column for term ID (must be unique), 2nd column for term name, 3rd column for term namespace, and 4th column for term distance. These four columns must be provided, but the content for the last column can be arbitrary (if it is hard to prepare)

`association.file`
: an input file containing associations between domains and terms. For example, a file containing associations 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 the header (in the first row) and two columns: 1st column for domain ID (corresponding to the first column in 'domain_info.file'), 2nd column for term ID (corresponding to the first column in 'term_info.file'). If there are additional columns, these columns will be ignored

`output.file`
: an output file used to save the built object as an RData-formatted file. If NULL, this file will be saved into "Anno.RData" in the current working local directory

Value

Any use-specified variable that is given on the right side of the assigement sign '<-', which contains the built Anno object. Also, an RData file specified in "output.file" is saved in the local directory.

Note

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

See Also

Anno

Examples

```
# build an "Anno" object that contains SCOP domain superfamilies (sf) annotated by GOBP terms
InterPro2GOMF <-
dcBuildAnno(domain_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt",
term_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/GO.txt",
association.file="http://dcgor.r-forge.r-project.org/data/InterPro/Domain2GOMF.txt",
output.file="InterPro2GOMF.RData")
InterPro2GOMF
```

---

dcBuildInfoDataFrame — *Function to build an object of the S4 class InfoDataframe from an input file*

Description

`dcBuildInfoDataFrame` is supposed to build an object of of the S4 class InfoDataFrame, given an input file. This input file can, for example, contain the domain information.

Usage

```
dcBuildInfoDataFrame(input.file, output.file = "InfoDataFrame.RData")
```

Arguments

input.file	an input file used to build the object. For example, a file containing InterPro domains (InterPro) can be found in http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt. As seen in this example, the input file must contain the header (in the first row), and entries in the first column intend to be domain identities (and must be unique)
output.file	an output file used to save the built object as an RData-formatted file. If NULL, this file will be saved into "InfoDataFrame.RData" in the current working local directory

Value

Any use-specified variable that is given on the right side of the assigement sign '<-', which contains the built `dcBuildInfoDataFrame` object. Also, an RData file specified in "output.file" is saved in the local directory.

Note

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

See Also

InfoDataFrame

Examples

```
# build an "InfoDataFrame" object that contains information on InterPro domains (InterPro)
InterPro <-
dcBuildInfoDataFrame(input.file="http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt",
output.file="InterPro.RData")
InterPro
```

`dcBuildOnto` *Function to build an object of the S4 class Onto from input files*

Description

`dcBuildOnto` is supposed to build an object of of the S4 class `Onto`, given input files. These input files include 1) a file containing term relations, and 2) a file containing term/node information.

Usage

```
dcBuildOnto(relations.file, nodes.file, output.file = "Onto.RData")
```

Arguments

`relations.file`	an input file containing term relations (i.e. edges from parent terms to child terms). For example, a file containing relations between GO Molecular Function (GOMF) terms can be found in `http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_edges.txt`. As seen in this example, the input file must contain the header (in the first row) and two columns: 1st column for parent term ID, and 2nd column for child term ID
`nodes.file`	an input file containing term/node information. For example, a file containing GO Molecular Function (GOMF) terms can be found in `http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_nodes.txt`. As seen in this example, the input file must contain the header (in the first row) and five columns: 1st column 'name' for node names (actually term ID; must be unique), 2nd column 'term_id' for term ID, 3rd 'term_name' for term name, 4th column 'term_namespace' for term namespace, and 5th column 'term_distance' for term distance. These five columns must be provided, the content in the first two columns are identical, and the content for the last column can be arbitrary (if it is hard to prepare)
`output.file`	an output file used to save the built object as an RData-formatted file. If NULL, this file will be saved into "Onto.RData" in the current working local directory

Value

Any use-specified variable that is given on the right side of the assigement sign '<-', which contains the built `Onto` object. Also, an RData file specified in "output.file" is saved in the local directory.

Note

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

See Also

Onto

Examples

```
# build an "Onto" object for GO Molecular Function
onto.GOMF <-
dcBuildOnto(relations.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_edges.txt",
nodes.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_nodes.txt",
output.file="onto.GOMF.RData")
onto.GOMF
```

---

dcConverter *Function to convert an object between graph classes*

Description

dcConverter is supposed to convert an object between classes 'Onto' and 'igraph', or between 'Dnetwork' and 'igraph', or between 'Cnetwork' and 'igraph'.

Usage

```
dcConverter(obj, from = c("Onto", "igraph", "Dnetwork", "Cnetwork"),
to = c("igraph", "Onto", "Dnetwork", "Cnetwork"), verbose = TRUE)
```

Arguments

obj	an object of class "Onto", "igraph", "Dnetwork" or "Cnetwork"
from	a character specifying the class converted from. It can be one of "Onto", "igraph", "Dnetwork" and "Dnetwork"
to	a character specifying the class converted to. It can be one of "Onto", "igraph", "Dnetwork" and "Dnetwork"
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 "Onto", "igraph", "Dnetwork" or "Cnetwork"

Note

Conversion is also supported between classes 'Onto' and 'igraph', or between 'Dnetwork' and 'igraph', or between 'Cnetwork' and 'igraph'

See Also

dcRDataLoader, Onto-class, Dnetwork-class, Cnetwork-class

Examples

```
# 1) conversion between Onto and igraph
# 1a) load onto.GOMF (as Onto object)
on <- dcRDataLoader(onto.GOMF)
on
# 1b) convert the object from Onto to igraph class
ig <- dcConverter(on, from=Onto, to=igraph)
ig
# 1c) convert the object from igraph to Onto class
dcConverter(ig, from=igraph, to=Onto)

# 2) conversion between Dnetwork and igraph
# 2a) computer a domain semantic network (as Dnetwork object)
g <- dcRDataLoader(onto.GOMF)
Anno <- dcRDataLoader(SCOP.sf2GOMF)
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,5) # randomly sample 5 domains
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=FALSE)
dnetwork
# 2b) convert the object from Dnetwork to igraph class
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
# 2c) convert the object from igraph to Dnetwork class
dcConverter(ig, from=igraph, to=Dnetwork)
```

---

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

Description

dcDAGannotate 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" or "Onto" object, 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) are also updated according to the true-path rule: a domain annotated to a term should also be annotated by its all ancestor terms.

Usage

```
dcDAGannotate(g, annotations, path.mode = c("all_paths",
"shortest_paths",
"all_shortest_paths"), verbose = TRUE)
```

Arguments

g	an object of class "igraph" or Onto

annotations	an object of class Anno, that is, the vertices/nodes for which annotation data are provided
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)
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" or "Onto" (the same as input). In addition to the original attributes to nodes and edges, the return subgraph is also appended by new node attributes: "annotations", which contains a list of domains either as original annotations or inherited annotations; "IC", which stands for information content defined as negative 10-based log-transformed frequency of domains 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

dcRDataLoader, dcEnrichment, dcDAGdomainSim, dcConverter

Examples

```
# 1) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)

# 2) load SCOP superfamilies annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(SCOP.sf2GOMF)

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

# 4) obtain the induced subgraph according to the input annotation data
# 4a) based on all possible paths (i.e. the complete subgraph induced)
dcDAGannotate(g, annotations, path.mode="all_paths", verbose=TRUE)
# 4b) based on shortest paths (i.e. the most concise subgraph induced)
dag <- dcDAGannotate(g, annotations, path.mode="shortest_paths",
verbose=TRUE)

# 5) color-code nodes/terms according to the number of annotations
if(class(dag)==Onto) dag <- dcConverter(dag, from=Onto,
to=igraph)
data <- sapply(V(dag)$annotations, length)
names(data) <- V(dag)$name
dnet::visDAG(g=dag, data=data, node.info="both")
```

dcDAGdomainSim — *Function to calculate pair-wise semantic similarity between domains based on a direct acyclic graph (DAG) with annotated data*

Description

dcDAGdomainSim is supposed to calculate pair-wise semantic similarity between domains based on a direct acyclic graph (DAG) with annotated data. It first calculates semantic similarity between terms and then derives semantic similarity between domains from terms-term semantic similarity. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
dcDAGdomainSim(g, domains = NULL, method.domain = c("BM.average",
"BM.max",
"BM.complete", "average", "max"), method.term = c("Resnik", "Lin",
"Schlicker", "Jiang", "Pesquita"), force = TRUE, fast = TRUE,
parallel = TRUE, multicores = NULL, verbose = TRUE)
```

Arguments

- `g` — an object of class "igraph" or Onto. It must contain a node attribute called 'annotations' for storing annotation data (see example for howto)
- `domains` — the domains between which pair-wise semantic similarity is calculated. If NULL, all domains annotatable in the input dag will be used for calcluation, which is very prohibitively expensive!
- `method.domain` — the method used for how to derive semantic similarity between domains from semantic similarity between terms. It can be "average" for average similarity between any two terms (one from domain 1, the other from domain 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 domain, first calculate maximum similarity to any term in the other domain, then take average of maximum similarity; the final BM-based average similiary is the pre-calculated average between two domains 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 domains 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 domain and a term of the other domain). When comparing BM-based similarity between domains, "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 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))

force
: logical to indicate whether the only most specific terms (for each domain) 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 dcDAGannotate)

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

an object of S4 class Dnetwork. It is a weighted and undirect graph, with following slots:

- `nodeInfo`: an object of S4 class, describing information on nodes/domains
- `adjMatrix`: an object of S4 class AdjData, containing symmetric adjacency data matrix for pair-wise semantic similarity between domains

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

dcRDataLoader, dcDAGannotate, dcConverter, Dnetwork-class

Examples

```
# 1) Semantic similarity between SCOP domain superfamilies (sf)
## 1a) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)
## 1b) load SCOP superfamilies annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(SCOP.sf2GOMF)
## 1c) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
## 1d) calculate pair-wise semantic similarity between 8 randomly chosen domains
```

```
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,8)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork
## 1e) convert it to an object of class igraph
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
## 1f) visualise the domain network
### extract edge weight (with 2-digit precision)
x <- signif(E(ig)$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
dnet::visNet(g=ig, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

###########################################################
# 2) Semantic similarity between Pfam domains (Pfam)
## 2a) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)
## 2b) load Pfam domains annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(Pfam2GOMF)
## 2c) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
## 2d) calculate pair-wise semantic similarity between 8 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,8)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork
## 2e) convert it to an object of class igraph
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
## 2f) visualise the domain network
### extract edge weight (with 2-digit precision)
x <- signif(E(ig)$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
dnet::visNet(g=ig, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

###########################################################
# 3) Semantic similarity between InterPro domains (InterPro)
## 3a) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)
## 3b) load InterPro domains annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(InterPro2GOMF)
## 3c) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
## 3d) calculate pair-wise semantic similarity between 8 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
```

```
domains <- sample(alldomains,8)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork
## 3e) convert it to an object of class igraph
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
## 3f) visualise the domain network
### extract edge weight (with 2-digit precision)
x <- signif(E(ig)$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
dnet::visNet(g=ig, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

###########################################################
# 4) Semantic similarity between Rfam RNA families (Rfam)
## 4a) load onto.GOBP (as Onto object)
g <- dcRDataLoader(onto.GOBP)
## 4b) load Rfam families annotated by GOBP (as Anno object)
Anno <- dcRDataLoader(Rfam2GOBP)
## 4c) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
## 4d) calculate pair-wise semantic similarity between 8 randomly chosen RNAs
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,8)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork
## 4e) convert it to an object of class igraph
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
## 4f) visualise the domain network
### extract edge weight (with 2-digit precision)
x <- signif(E(ig)$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
dnet::visNet(g=ig, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)

###########################################################
# 5) Advanced usage: customised data for ontology and annotations
# 5a) customise ontology
g <-
dcBuildOnto(relations.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_edges.txt",
nodes.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_nodes.txt",
output.file="ontology.RData")
# 5b) customise Anno
Anno <-
dcBuildAnno(domain_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt",
term_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/GO.txt",
association.file="http://dcgor.r-forge.r-project.org/data/InterPro/Domain2GOMF.txt",
```

```
output.file="annotations.RData")
## 5c) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=FALSE)
## 5d) calculate pair-wise semantic similarity between 8 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,8)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork
## 5e) convert it to an object of class igraph
ig <- dcConverter(dnetwork, from=Dnetwork, to=igraph)
ig
## 5f) visualise the domain network
### extract edge weight (with 2-digit precision)
x <- signif(E(ig)$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
dnet::visNet(g=ig, vertex.shape="sphere", edge.width=edge.width,
edge.label=x, edge.label.cex=0.7)
```

`dcDuplicated` *Function to determine the duplicated patterns from input data matrix*

Description

`dcDuplicated` is supposed to determine the duplicated vectorised patterns from a matrix or data frame. The patterns can come from column-wise vectors or row-wise vectors. It returns an integer vector, in which the value indicates from which it duplicats.

Usage

```
dcDuplicated(data, pattern.wise = c("column", "row"), verbose = T)
```

Arguments

`data`	an input data matrix/frame
`pattern.wise`	a character specifying in which wise to define patterns from input data. It can be 'column' for column-wise vectors, and 'row' for row-wise vectors
`verbose`	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display

Value

an interger vector, in which an entry indicates from which it duplicats. When viewing column-wise patterns (or row-wise patterns), the returned integer vector has the same length as the column number (or the row number) of input data.

Note

none

See Also

dcAncestralMP, dcAncestralMP, dcAlgo

Examples

```
# an input data matrix storing discrete states for tips (in rows) X four characters (in columns)
data1 <- matrix(c(0,rep(1,3),rep(0,2)), ncol=1)
data2 <- matrix(c(rep(0,4),rep(1,2)), ncol=1)
data3 <- matrix(c(1,rep(0,3),rep(1,2)), ncol=1)
data <- cbind(data1, data2, data1, data3)
colnames(data) <- c("C1", "C2", "C3", "C4")
data

# determine the duplicated patterns from inut data matrix
res <- dcDuplicated(data, pattern.wise="column")
## return an integer vector
res
## get index for unique patterns
ind <- sort(unique(res))
## As seen above, the returned integer vector tells there are 3 unique patterns:
## they are in columns (1, 2, 4). The column 3 is duplicated from column 1.
```

---

dcEnrichment *Function to conduct ontology enrichment analysis given a group of domains*

Description

dcEnrichment is supposed to conduct enrichment analysis for an input group of domains using a specified ontology. It returns an object of S4 class "Eoutput". Enrichment analysis is based on either Fisher's exact test or Hypergeometric test. The test can respect the hierarchy of the ontology. The user can customise the background domains; otherwise, the function will use all annotatable domains as the test background

Usage

```
dcEnrichment(data, background = NULL, domain = c(NA, "SCOP.sf",
"SCOP.fa",
"Pfam", "InterPro", "Rfam"), ontology = c(NA, "GOBP", "GOMF", "GOCC",
"DO",
"HPPA", "HPMI", "HPON", "MP", "EC", "KW", "UP"), sizeRange = c(10,
1000),
min.overlap = 3, which_distance = NULL, test = c("HypergeoTest",
"FisherTest", "BinomialTest"), p.adjust.method = c("BH", "BY",
"bonferroni",
"holm", "hochberg", "hommel"), ontology.algorithm = c("none", "pc",
"elim",
"lea"), elim.pvalue = 0.01, lea.depth = 2, verbose = T,
domain.RData = NULL, ontology.RData = NULL, annotations.RData = NULL,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

`data`	an input vector. It contains id for a list of domains, for example, sunids for SCOP domains
`background`	a background vector. It contains id for a list of background domains, for example, sunids for SCOP domains. If NULL, by default all annotatable domains are used as background
`domain`	the domain identity. It can be one of 'SCOP.sf' for SCOP superfamilies, 'SCOP.fa' for SCOP families, 'Pfam' for Pfam domains, 'InterPro' for InterPro domains, 'Rfam' for Rfam RNA families
`ontology`	the ontology identity. 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, "HPON" for Human Phenotype ONset and clinical course, "MP" for Mammalian Phenotype, "EC" for Enzyme Commission, "KW" for UniProtKB KeyWords, "UP" for UniProtKB UniPathway. For details on the eligibility for pairs of input domain and ontology, please refer to the online Documentations at `http://supfam.org/dcGOR/docs.html`
`sizeRange`	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 1000
`min.overlap`	the minimum number of overlaps. Only those terms that overlap with input data at least min.overlap (3 domains by default) will be processed
`which_distance`	which distance of terms in the ontology 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 "FisherTest" for using fisher's exact test, "HypergeoTest" for using hypergeometric test, or "BinomialTest" for using binomial test. Fisher's exact test is to test the independence between domain group (domains belonging to a group or not) and domain annotation (domains 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 domains, 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'
`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 domains 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 uded to consider the children of a term (and subsequently all domains in these children term are eliminated from the use for the recalculation of the signifance at this term)

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

`domain.RData` a file name for RData-formatted file containing an object of S4 class 'InfoDataFrame' (i.g. domain). By default, it is NULL. It is only needed when the user wants to customise enrichment analysis using their own data. See `dcBuildInfoDataFrame` for how to creat this object

`ontology.RData` a file name for RData-formatted file containing an object of S4 class 'Onto' (i.g. ontology). By default, it is NULL. It is only needed when the user wants to customise enrichment analysis using their own data. See `dcBuildOnto` for how to creat this object

`annotations.RData`
a file name for RData-formatted file containing an object of S4 class 'Anno' (i.g. annotations). By default, it is NULL. It is only needed when the user wants to customise enrichment analysis using their own data. See `dcBuildAnno` for how to creat this object

`RData.location` the characters to tell the location of built-in RData files. By default, it remotely locates at "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

Value

an object of S4 class `Eoutput`, with following slots:

- `domain`: a character specifying the domain identity
- `ontology`: a character specifying the ontology used
- `term_info`: a matrix of nTerm X 5 containing term information, where nTerm is the number of terms in consideration, and the 5 columns are "term_id" (i.e. "Term ID"), "term_name" (i.e. "Term Name"), "namespace" (i.e. "Term Namespace"), "distance" (i.e. "Term Distance") and "IC" (i.e. "Information Content for the term based on annotation frequency by it")
- `anno`: a list of terms, each storing annotated domain members (also within the background domains). Always, terms are identified by "term_id" and domain members identified by their ids (e.g. sunids for SCOP domains)
- `data`: a vector containing input data in consideration. It is not always the same as the input data as only those mappable and annotatable are retained
- `background`: a vector containing background in consideration. It is not always the same as the input background as only those mappable/annotatable are retained
- `overlap`: a list of terms, each storing domains overlapped between domains annotated by a term and domains in the input data (i.e. the domains of interest). Always, terms are identified by "term_id" and domain members identified by their IDs (e.g. sunids for SCOP domains)

- 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

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 domains are already annotated to any children terms with a more signficance than itself, then all these domains 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 domains are already annotated to a signficantly enriched term under the cutoff of e.g. pvalue<1e-2, all these domains are eliminated from the ancestors of that term).
- "pc": requires the significance of a term not only using the whole domains as background but also using domains 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

dcRDataLoader, dcDAGannotate, Eoutput-class, visEnrichment, dcConverter

Examples

```
# 1) Enrichment analysis for SCOP domain superfamilies (sf)
## 1a) load SCOP.sf (as InfoDataFrame object)
SCOP.sf <- dcRDataLoader(SCOP.sf)
### randomly select 50 domains as a list of domains of interest
data <- sample(rowNames(SCOP.sf), 50)
## 1b) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="SCOP.sf", ontology="GOMF")
eoutput
## 1c) view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
## 1d) visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
## 1e) the same as above but using a customised background
### randomly select 500 domains as background
background <- sample(rowNames(SCOP.sf), 500)
### perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, background=background, domain="SCOP.sf",
ontology="GOMF")
eoutput
### view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
### visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
```

```
###########################################################
# 2) Enrichment analysis for Pfam domains (Pfam)
## 2a) load Pfam (as InfoDataFrame object)
Pfam <- dcRDataLoader(Pfam)
### randomly select 100 domains as a list of domains of interest
data <- sample(rowNames(Pfam), 100)
## 2b) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="Pfam", ontology="GOMF")
eoutput
## 2c) view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
## 2d) visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
## 2e) the same as above but using a customised background
### randomly select 1000 domains as background
background <- sample(rowNames(Pfam), 1000)
### perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, background=background, domain="Pfam",
ontology="GOMF")
eoutput
### view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
### visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)

###########################################################
# 3) Enrichment analysis for InterPro domains (InterPro)
## 3a) load InterPro (as InfoDataFrame object)
InterPro <- dcRDataLoader(InterPro)
### randomly select 100 domains as a list of domains of interest
data <- sample(rowNames(InterPro), 100)
## 3b) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="InterPro", ontology="GOMF")
eoutput
## 3c) view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
## 3d) visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
## 3e) the same as above but using a customised background
### randomly select 1000 domains as background
background <- sample(rowNames(InterPro), 1000)
### perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, background=background, domain="InterPro",
ontology="GOMF")
eoutput
### view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
### visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)

###########################################################
# 4) Enrichment analysis for Rfam RNA families (Rfam)
```

```
## 4a) load Rfam (as InfoDataFrame object)
Rfam <- dcRDataLoader(Rfam)
### randomly select 100 RNAs as a list of RNAs of interest
data <- sample(rowNames(Rfam), 100)
## 4b) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="Rfam", ontology="GOBP")
eoutput
## 4c) view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=FALSE)
## 4d) visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
## 4e) the same as above but using a customised background
### randomly select 1000 RNAs as background
background <- sample(rowNames(Rfam), 1000)
### perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, background=background, domain="Rfam",
ontology="GOBP")
eoutput
### view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=FALSE)
### visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)

###########################################################
# 5) Advanced usage: customised data for domain, ontology and annotations
# 5a) create domain, ontology and annotations
## for domain
domain <-
dcBuildInfoDataFrame(input.file="http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt",
output.file="domain.RData")
## for ontology
dcBuildOnto(relations.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_edges.txt",
nodes.file="http://dcgor.r-forge.r-project.org/data/onto/igraph_GOMF_nodes.txt",
output.file="ontology.RData")
## for annotations
dcBuildAnno(domain_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/InterPro.txt",
term_info.file="http://dcgor.r-forge.r-project.org/data/InterPro/GO.txt",
association.file="http://dcgor.r-forge.r-project.org/data/InterPro/Domain2GOMF.txt",
output.file="annotations.RData")
## 5b) prepare data and background
### randomly select 100 domains as a list of domains of interest
data <- sample(rowNames(domain), 100)
### randomly select 1000 domains as background
background <- sample(rowNames(domain), 1000)
## 5c) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, background=background,
domain.RData=domain.RData, ontology.RData=ontology.RData,
annotations.RData=annotations.RData)
eoutput
## 5d) view the top 10 significance terms
view(eoutput, top_num=10, sortBy="pvalue", details=TRUE)
### visualise the top 10 significant terms in the ontology hierarchy
### color-coded according to 10-based negative logarithm of adjusted p-values (adjp)
visEnrichment(eoutput)
```

dcRDataLoader *Function to load dcGOR built-in RData*

Description

dcRDataLoader is supposed to load RData that are used by package dcGOR.

Usage

```
dcRDataLoader(RData = c(NA, "SCOP.sf", "SCOP.fa", "Pfam", "InterPro",
"Rfam",
"onto.GOBP", "onto.GOMF", "onto.GOCC", "onto.DO", "onto.HPPA",
"onto.HPMI",
"onto.HPON", "onto.MP", "onto.EC", "onto.KW", "onto.UP",
"SCOP.sf2GOBP",
"SCOP.sf2GOMF", "SCOP.sf2GOCC", "SCOP.sf2DO", "SCOP.sf2HPPA",
"SCOP.sf2HPMI",
"SCOP.sf2HPON", "SCOP.sf2MP", "SCOP.sf2EC", "SCOP.sf2KW", "SCOP.sf2UP",
"SCOP.fa2GOBP", "SCOP.fa2GOMF", "SCOP.fa2GOCC", "SCOP.fa2DO",
"SCOP.fa2HPPA",
"SCOP.fa2HPMI", "SCOP.fa2HPON", "SCOP.fa2MP", "SCOP.fa2EC",
"SCOP.fa2KW",
"SCOP.fa2UP", "Pfam2GOBP", "Pfam2GOMF", "Pfam2GOCC", "InterPro2GOBP",
"InterPro2GOMF", "InterPro2GOCC", "Rfam2GOBP", "Rfam2GOMF",
"Rfam2GOCC",
"Ancestral_domainome", "eTOL", "Feature2GOBP.sf", "Feature2GOMF.sf",
"Feature2GOCC.sf", "Feature2HPPA.sf", "Feature2GOBP.pfam",
"Feature2GOMF.pfam", "Feature2GOCC.pfam", "Feature2HPPA.pfam",
"Feature2GOBP.interpro", "Feature2GOMF.interpro",
"Feature2GOCC.interpro",
"Feature2HPPA.interpro"), domain = c(NA, "SCOP.sf", "SCOP.fa", "Pfam",
"InterPro", "Rfam"), ontology = c(NA, "GOBP", "GOMF", "GOCC", "DO",
"HPPA",
"HPMI", "HPON", "MP", "EC", "KW", "UP"), verbose = T,
RData.location = "http://dcgor.r-forge.r-project.org/data")
```

Arguments

RData	which built-in RData to load. If NOT NA, this RData will be always loaded. It can be: domains/RNAs (including 'SCOP.sf', 'SCOP.fa', 'Pfam', 'InterPro', 'Rfam'), ontologies (including 'onto.GOBP', 'onto.GOMF', 'onto.GOCC', 'onto.DO', 'onto.HPPA', 'onto.HPMI', 'onto.HPON', 'onto.MP', 'onto.EC', 'onto.KW', 'onto.UP'), annotations (including 'SCOP.sf2GOBP', 'SCOP.sf2GOMF', 'SCOP.sf2GOCC', 'SCOP.sf2DO', 'SCOP.sf2HPPA', 'SCOP.sf2HPMI', 'SCOP.sf2HPON', 'SCOP.sf2MP', 'SCOP.sf2EC', 'SCOP.sf2KW', 'SCOP.sf2UP', 'SCOP.fa2GOBP', 'SCOP.fa2GOMF', 'SCOP.fa2GOCC', 'SCOP.fa2DO', 'SCOP.fa2HPPA', 'SCOP.fa2HPMI', 'SCOP.fa2HPON', 'SCOP.fa2MP', 'SCOP.fa2EC', 'SCOP.fa2KW', 'SCOP.fa2UP', 'Pfam2GOBP', 'Pfam2GOMF', 'Pfam2GOCC', 'InterPro2GOBP', 'InterPro2GOMF', 'InterPro2GOCC', 'Rfam2GOBP', 'Rfam2GOMF', 'Rfam2GOCC'), domainome in eukaryotic genomes (including 'Ancestral_domainome', 'eTOL'), and databases used for predictiing ontology terms from input protein domain contents. On the meanings, please refer to the Documentations

`domain` domain part of annotation RData to load. When RData is NA and this plus next are NOT NA, then this plus next one are used to specify which annotation RData to load. In addition to NA, it can also be: 'SCOP.sf', 'SCOP.fa', 'Pfam' and 'InterPro'

`ontology` ontology part of annotation RData to load. This only works together with the previous 'domain' parameter. In addition to NA, it can also be: 'GOBP', 'GOMF', 'GOCC', 'DO', 'HPPA', 'HPMI', 'HPON', 'MP', 'EC', 'KW', 'UP'

`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 "http://supfam.org/dcGOR/data" or "http://dcgor.r-forge.r-project.org/data". 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: $RData.location = "."$. If RData to load is already part of package itself, this parameter can be ignored (since this function will try to load it via function `data` first)

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

dcEnrichment

Examples

```
# Always, load from specified RData directly
SCOP.sf <- dcRDataLoader(RData=SCOP.sf)
Pfam <- dcRDataLoader(RData=Pfam)
InterPro <- dcRDataLoader(RData=InterPro)
Rfam <- dcRDataLoader(RData=Rfam)
onto.GOMF <- dcRDataLoader(RData=onto.GOMF)
# But for annotaion data, there are two ways to do so:
# 1) in a direct way
SCOP.sf2GOMF <- dcRDataLoader(RData=SCOP.sf2GOMF)
# 2) in an indirect way: specify both domain and ontology
SCOP.sf2GOMF <- dcRDataLoader(domain=SCOP.sf, ontology=GOMF)
```

dcRWRpipeline	*Function to setup a pipeine to estimate RWR-based contact strength between samples from an input domain-sample data matrix and an input graph*

Description

dcRWRpipeline is supposed to estimate sample relationships (ie. contact strength between samples) from an input domain-sample matrix and an input graph (such as a domain-domain semantic network). The pipeline includes: 1) random walk restart (RWR) of the input graph using the input matrix as seeds; 2) calculation of contact strength (inner products of RWR-smoothed columns of input matrix); 3) estimation of the contact signficance by a randomalisation procedure. It supports two methods how to use RWR: 'direct' for directly applying RWR in the given seeds; 'indirectly' for first pre-computing affinity matrix of the input graph, and then deriving the affinity score. Parallel computing is also supported for Linux or Mac operating systems.

Usage

```
dcRWRpipeline(data, g, method = c("indirect", "direct"),
normalise = c("laplacian", "row", "column", "none"), restart = 0.75,
normalise.affinity.matrix = c("none", "quantile"),
permutation = c("random", "degree"), num.permutation = 100,
p.adjust.method = c("BH", "BY", "bonferroni", "holm", "hochberg",
"hommel"),
adjp.cutoff = 0.05, parallel = TRUE, multicores = NULL, verbose = T)
```

Arguments

- `data` an input domain-sample data matrix used for seeds. Each value in input domain-sample matrix does not necessarily have to be binary (non-zeros will be used as a weight, but should be non-negative for easy interpretation).
- `g` an object of class "igraph" or Dnetwork
- `method` the method used to calculate RWR. It can be 'direct' for directly applying RWR, 'indirect' for indirectly applying RWR (first pre-compute affinity matrix and then derive the affinity score)
- `normalise` the way to normalise the adjacency matrix of the input graph. It can be 'laplacian' for laplacian normalisation, 'row' for row-wise normalisation, 'column' for column-wise normalisation, or 'none'
- `restart` the restart probability used for RWR. The restart probability takes the value from 0 to 1, controlling the range from the starting nodes/seeds that the walker will explore. The higher the value, the more likely the walker is to visit the nodes centered on the starting nodes. At the extreme when the restart probability is zero, the walker moves freely to the neighbors at each step without restarting from seeds, i.e., following a random walk (RW)
- `normalise.affinity.matrix` the way to normalise the output affinity matrix. It can be 'none' for no normalisation, 'quantile' for quantile normalisation to ensure that columns (if multiple) of the output affinity matrix have the same quantiles
- `permutation` how to do permutation. It can be 'degree' for degree-preserving permutation, 'random' for permutation in random

`num.permutation`
: the number of permutations used to for generating the distribution of contact strength under randomalisation

`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

`adjp.cutoff`
: the cutoff of adjusted pvalue to construct the contact graph

`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

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

- `ratio`: a symmetric matrix storing ratio (the observed against the expected) between pairwise samples
- `zscore`: a symmetric matrix storing zscore between pairwise samples
- `pval`: a symmetric matrix storing pvalue between pairwise samples
- `adjpval`: a symmetric matrix storing adjusted pvalue between pairwise samples
- `icontact`: the constructed contact graph (as an 'igraph' object) under the cutoff of adjusted value
- `Amatrix`: a pre-computated affinity matrix when using 'inderect' method; NULL otherwise
- `call`: the call that produced this result

Note

The choice of which method to use RWR depends on the number of seed sets and the number of permutations for statistical test. If the total product of both numbers are huge, it is better to use 'indrect' method (for a single run).

See Also

`dcRDataLoader`, `dcDAGannotate`, `dcDAGdomainSim`, `dcConverter`

Examples

```
# 1) load onto.GOMF (as Onto object)
g <- dcRDataLoader(onto.GOMF)

# 2) load SCOP superfamilies annotated by GOMF (as Anno object)
Anno <- dcRDataLoader(SCOP.sf2GOMF)

# 3) prepare for ontology appended with annotation information
dag <- dcDAGannotate(g, annotations=Anno, path.mode="shortest_paths",
verbose=TRUE)

# 4) calculate pair-wise semantic similarity between 10 randomly chosen domains
alldomains <- unique(unlist(nInfo(dag)$annotations))
domains <- sample(alldomains,10)
dnetwork <- dcDAGdomainSim(g=dag, domains=domains,
method.domain="BM.average", method.term="Resnik", parallel=FALSE,
verbose=TRUE)
dnetwork

# 5) estimate RWR dating based sample/term relationships
# define sets of seeds as data
# each seed with equal weight (i.e. all non-zero entries are 1)
data <- data.frame(aSeeds=c(1,0,1,0,1), bSeeds=c(0,0,1,0,1))
rownames(data) <- id(dnetwork)[1:5]
# calcualte their two contact graph
coutput <- dcRWRpipeline(data=data, g=dnetwork, parallel=FALSE)
coutput
```

---

dcSplitArch *Function to obtain a list of features via splitting an input architecture*

Description

dcSplitArch is supposed to obtain a list of features via splitting an input architecture.

Usage

```
dcSplitArch(da, feature.mode = c("supra", "individual", "comb"), sep =
",",
ignore = "_gap_", verbose = T)
```

Arguments

da	an input architecture. For example, a comma-separated string
feature.mode	the mode of how to define the features thereof. It can be: "supra" for combinations of one or two successive domains (including individual domains; considering the order), "individual" for individual domains only, and "comb" for all possible combinations (including individual domains; ignoring the order)
sep	a character string to separate. By default, it is comma ','
ignore	a character string to ignore. By default, it is '_gap_'. Ihis ignored character will affect the features defined as being 'supra' (see examples below)
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display

Value

an interger vector, in which an entry indicates from which it duplicats. When viewing column-wise patterns (or row-wise patterns), the returned integer vector has the same length as the column number (or the row number) of input data.

Note

none

See Also

dcAlgo, dcAlgoPredict

Examples

```
da <- "_gap_,100895,57610,_gap_,57610,47473"
# get features defined as being "supra"
dcSplitArch(da, feature.mode="supra")
# get features defined as being "individual"
dcSplitArch(da, feature.mode="individual")
# get features defined as being "comb"
dcSplitArch(da, feature.mode="comb")
```

---

dcSubtreeClade *Function to extract a subtree under a given clade from a phylo-formatted phylogenetic tree*

Description

dcSubtreeClade is supposed to extract a subtree under a given clade from a phylo-formatted phylogenetic tree. In addition to the tree in subject, another input is a built-in integer specifying an internal node/clade of interest. Alternatively, the internal node of interest can be given by its label (if there are internal node labels). As a result, a subtree under a given clade is also represented as an object of class 'phylo'.

Usage

```
dcSubtreeClade(phy, choose.node = NULL, choose.node.label = NULL,
verbose = T)
```

Arguments

phy	an object of class 'phylo'
choose.node	an integer specifying which internal node is chosen. For an object of class 'phylo', the tree has built-in ID for internal nodes, ranging from $Ntip + 1$ to $Ntip + Nnode$, where $Ntip$ and $Nnode$ are the number of tips and internal nodes. Internal nodes are indexed in a pre-ordered manner. The subtree under the given interna node will be extracted
choose.node.label	a character specifying which internal node is chosen. For the tree with internal node labels, the extraction of subtree can be done in this way
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 'phylo'

Note

If a valid 'choose.node' is given, then 'choose.node.label' will be ignored.

See Also

dcTreeConnectivity

Examples

```
# 1) a newick tree without internal node labels
tree <- "(((t1:5,t2:5):2,(t3:4,t4:4):3):2,(t5:4,t6:4):6);"
phy <- ape::read.tree(text=tree)
phy
Ntip <- ape::Ntip(phy)
Nnode <- ape::Nnode(phy)
ape::plot.phylo(phy, type="p", use.edge.length=TRUE)
ape::nodelabels(node=Ntip+1:Nnode, col="red", bg="white")
# a subtree specified via a built-in internal node ID
subphy <- dcSubtreeClade(phy, choose.node=Ntip+2)
subphy
ape::plot.phylo(subphy, type="p", use.edge.length=TRUE)

# 2) a newick tree with internal node labels
tree <- "(((t1:5,t2:5)i3:2,(t3:4,t4:4)i4:3)i2:2,(t5:4,t6:4)i5:6)i1;"
phy <- ape::read.tree(text=tree)
phy
ape::plot.phylo(phy, type="p", use.edge.length=TRUE,
show.node.label=TRUE)
# a subtree specified via an internal node label
subphy <- dcSubtreeClade(phy, choose.node.label=i2)
subphy
ape::plot.phylo(subphy, type="p", use.edge.length=TRUE,
show.node.label=TRUE)
```

---

dcSubtreeTips *Function to extract a tip-induced subtree from a phylo-formatted phylogenetic tree*

Description

dcSubtreeTips is supposed to extract a tip-induced subtree from a phylo-formatted phylogenetic tree. In addition to the tree in subject, another input is a vector containing tip labels of interest. From valid tip lables, there are two types of subtree to extract. One is first induce clade (an internal node) from tip labels, and then the subtree is extracted under the induced clade. Another type is to extract a subtree only containing given tip labels; in this situation, some internal nodes perhaps need to further trimmed. The resulting subtree is also represented as an object of class 'phylo'.

Usage

```
dcSubtreeTips(phy, choose.tip.labels = NULL, subtree.type = c("clade",
"tips_only"), verbose = T)
```

Arguments

phy	an object of class 'phylo'
choose.tip.labels	a character specifying which tips are chosen
subtree.type	a character specifying how to extract subtree from given tips. It can be 'clade' or 'tips_only'. The former is first induce clade (an internal node) from tip labels, and then to extract the subtree under the induced clade. The latter is to directly extract the subtree only containing given tip labels, (if necessary), after trimming out unnecessary internal nodes
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 'phylo'

Note

nonde

See Also

dcTreeConnectivity, dcSubtreeClade

Examples

```
# 1) with internal node labels
tree <- "(((t1:5,t2:5)i3:2,(t3:4,t4:4)i4:3)i2:2,(t5:4,t6:4)i5:6)i1;"
phy <- ape::read.tree(text=tree)
ape::plot.phylo(phy, type="p", use.edge.length=TRUE,
show.node.label=TRUE)

# 2) tip labels of interest
choose.tip.labels <- c(t1,t2,t3)
# 2a) extract subtree via an induced clade
subphy <- dcSubtreeTips(phy, choose.tip.labels, subtree.type="clade")
ape::plot.phylo(subphy, type="p", use.edge.length=TRUE,
show.node.label=TRUE)
# 2b) extract subtree containing only tips
subphy <- dcSubtreeTips(phy, choose.tip.labels,
subtree.type="tips_only")
ape::plot.phylo(subphy, type="p", use.edge.length=TRUE,
show.node.label=TRUE)
```

dcTreeConnectivity *Function to calculate the sparse connectivity matrix between parents and children from a phylo-formatted phylogenetic tree*

Description

dcTreeConnectivity is supposed to calculate the sparse connectivity matrix between parents and children from a phylo-formatted phylogenetic tree. The matrix has internal nodes (in rows) and tips plus internal nodes (in columns). For a row (an internal node; as a parent), the non-zeros indicate all its descendants/children.

Usage

```
dcTreeConnectivity(phy, verbose = T)
```

Arguments

phy	an object of class 'phylo'
verbose	logical to indicate whether the messages will be displayed in the screen. By default, it sets to TRUE for display

Value

a sparse matrix of $Nnode$ X $Ntip + Nnode$, where $Ntip$ and $Nnode$ are the number of tips and internal nodes. A non-zero entry indicates a pair of a parent and its child.

Note

None

See Also

dcTreeConnectivity

Examples

```
# a newick tree
tree <- "(((t1:5,t2:5):2,(t3:4,t4:4):3):2,(t5:4,t6:4):6);"
phy <- ape::read.tree(text=tree)

# connectivity matrix
res <- dcTreeConnectivity(phy)
dim(res)
# convert to a full Matrix
as.matrix(res)
```

Dnetwork-class *Definition for S4 class Dnetwork*

Description

Dnetwork is an S4 class to store a domain network, such as the one from semantic similairty between pairs of domains by dcDAGdomainSim. It has 2 slots: nodeInfo and adjMatrix

Value

Class Dnetwork

Slots

nodeInfo An object of S4 class InfoDataFrame, describing information on nodes/domains.

adjMatrix An object of S4 class AdjData, containing symmetric adjacency data matrix for an indirect domain network

Creation

An object of this class can be created via: `new("Dnetwork",nodeInfo, adjMatrix)`

Methods

Class-specific methods:

- `dim()`: retrieve the dimension in the object
- `adjMatrix()`: retrieve the slot 'adjMatrix' in the object
- `nodeInfo()`: retrieve the slot 'nodeInfo' (as class InfoDataFrame) in the object
- `nInfo()`: retrieve nodeInfo (as data.frame) in the object
- `nodeNames()`: retrieve node/term names (ie, row names of nodeInfo) in the object
- `id()`: retrieve domain id (ie, column 'id' of nodeInfo) in the object, if any
- `level()`: retrieve domain level (ie, column 'level' of nodeInfo) in the object, if any
- `description()`: retrieve domain description (ie, column 'description' of nodeInfo) in the object, if any

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object
- `as(matrix, "Dnetwork")`: convert a matrix to an object of class Dnetwork
- `as(dgCMatrix, "Dnetwork")`: convert a dgCMatrix (a sparse matrix) to an object of class Dnetwork
- `[i]`: get the subset of the same class

Access

Ways to access information on this class:

- `showClass("Dnetwork")`: show the class definition
- `showMethods(classes="Dnetwork")`: show the method definition upon this class
- `getSlots("Dnetwork")`: get the name and class of each slot in this class
- `slotNames("Dnetwork")`: get the name of each slot in this class
- `selectMethod(f, signature="Dnetwork")`: retrieve the definition code for the method 'f' defined in this class

See Also

Dnetwork-method

Examples

```
# create an object of class Dnetwork, only given a matrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
as(adjM, "Dnetwork")

# create an object of class Dnetwork, given a matrix plus information on nodes
# 1) create nodeI: an object of class InfoDataFrame
data <- data.frame(id=paste("Domain", 1:5, sep="_"),
level=rep("SCOP",5), description=I(LETTERS[1:5]),
row.names=paste("Domain", 1:5, sep="_"))
nodeI <- new("InfoDataFrame", data=data)
nodeI
# 2) create an object of class Dnetwork
# VERY IMPORTANT: make sure having consistent names between nodeInfo and adjMatrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
colnames(adjM) <- rownames(adjM) <- rowNames(nodeI)
x <- new("Dnetwork", adjMatrix=adjM, nodeInfo=nodeI)
x
# 3) look at various methods defined on class Dnetwork
dim(x)
adjMatrix(x)
nodeInfo(x)
nInfo(x)
nodeNames(x)
id(x)
level(x)
description(x)
# 4) get the subset
x[1:2]
```

Dnetwork-method *Methods defined for S4 class Dnetwork*

Description

Methods defined for class `Dnetwork`.

Usage

```
## S4 method for signature Dnetwork
dim(x)

## S4 method for signature Dnetwork
adjMatrix(x)

## S4 method for signature Dnetwork
nodeInfo(x)

## S4 method for signature Dnetwork
nInfo(object)

## S4 method for signature Dnetwork
nodeNames(object)

## S4 method for signature Dnetwork
id(object)

## S4 method for signature Dnetwork
level(object)

## S4 method for signature Dnetwork
description(object)

## S4 method for signature Dnetwork
show(object)

## S4 method for signature Dnetwork,ANY,ANY,ANY
x[i, j, ..., drop = FALSE]
```

Arguments

`x`	an object of class `Dnetwork`
`object`	an object of class `Dnetwork`
`i`	an index
`j`	an index
`...`	additional parameters
`drop`	a logic for matrices and arrays. If TRUE the result is coerced to the lowest possible dimension. This only works for extracting elements, not for the replacement

See Also

`Dnetwork-class`

Eoutput-class *Definition for S4 class Eoutput*

Description

`Eoutput` is an S4 class to store output from enrichment analysis by `dcEnrichment`.

Value

Class Eoutput

Slots

`domain` A character specifying the domain identity

`ontology` A character specifying the ontology identity

`term_info` A data.frame of nTerm X 5 containing term information, where nTerm is the number of terms in consideration, and the 5 columns are "term_id" (i.e. "Term ID"), "term_name" (i.e. "Term Name"), "namespace" (i.e. "Term Namespace"), "distance" (i.e. "Term Distance") and "IC" (i.e. "Information Content for the term based on annotation frequency by it")

`anno` A list of terms, each storing annotated domains (also within the background domains). Always, terms are identified by "term_id" and domain members identified by their ids (e.g. sunids for SCOP domains)

`data` A vector containing input data in `dcEnrichment`. It is not always the same as the input data as only those mappable are retained

`background` A vector containing background in `dcEnrichment`. It is not always the same as the input background (if provided by the user) as only those mappable are retained

`overlap` A list of terms, each storing domains overlapped between domains annotated by a term and domains in the input data (i.e. the domains of interest). Always, terms are identified by "term_id" and domain members identified by their ids (e.g. sunids for SCOP domains)

`zscore` A vector of terms, containing z-scores

`pvalue` A vector of terms, containing p-values

`adjp` A vector of terms, containing adjusted p-values. It is the p value but after being adjusted for multiple comparisons

Creation

An object of this class can be created via: `new("Eoutput",domain, ontology, term_info, anno, data, overlap,`

Methods

Class-specific methods:

- `zscore()`: retrieve the slot 'zscore' in the object
- `pvalue()`: retrieve the slot 'pvalue' in the object
- `adjp()`: retrieve the slot 'adjp' in the object
- `view()`: retrieve an integrated data.frame used for viewing the object
- `write()`: write the object into a local file

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object

Access

Ways to access information on this class:

- `showClass("Eoutput")`: show the class definition
- `showMethods(classes="Eoutput")`: show the method definition upon this class
- `getSlots("Eoutput")`: get the name and class of each slot in this class
- `slotNames("Eoutput")`: get the name of each slot in this class
- `selectMethod(f, signature="Eoutput")`: retrieve the definition code for the method 'f' defined in this class

See Also

Eoutput-method

Examples

```
# 1) load SCOP.sf (as InfoDataFrame object)
SCOP.sf <- dcRDataLoader(SCOP.sf)
# randomly select 20 domains
data <- sample(rowNames(SCOP.sf), 20)

# 2) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="SCOP.sf", ontology="GOMF")
eoutput

# 3) write into the file Eoutput.txt in your local directory
write(eoutput, file=Eoutput.txt)

# 4) view the top 5 significant terms
view(eoutput, top_num=5, sortBy="pvalue", details=TRUE)

# 4) retrieve several slots directly
zscore(eoutput)[1:5]
pvalue(eoutput)[1:5]
adjp(eoutput)[1:5]
```

`Eoutput-method` *Methods defined for S4 class Eoutput*

Description

Methods defined for S4 class `Eoutput`.

Usage

```
## S4 method for signature Eoutput
show(object)

## S4 method for signature Eoutput
zscore(x)

## S4 method for signature Eoutput
pvalue(x)

## S4 method for signature Eoutput
adjp(x)

## S4 method for signature Eoutput
view(x, top_num = 5, sortBy = c("pvalue", "adjp",
"zscore", "nAnno", "nOverlap", "none"), decreasing = NULL, details = T)

## S4 method for signature Eoutput
write(x, file = "Eoutput.txt", verbose = T)
```

Arguments

`object`	an object of S4 class `Eoutput`. Usually this is an output from `dcEnrichment`
`x`	an object of S4 class `Eoutput`. Usually this is an output from `dcEnrichment`
`top_num`	the maximum number (5, by default) of terms will be viewed. If NULL or NA, all terms will be viewed (this can be used for the subsequent saving)
`sortBy`	which statistics will be used for sorting and viewing terms. It can be "pvalue" for p value, "adjp" for adjusted p value, "zscore" for enrichment z-score, "nAnno" for the number in domains annotated by a term, "nOverlap" for the number in overlaps, and "none" for ordering simply according to ID of terms
`decreasing`	logical to indicate whether to sort in a decreasing order. If it is null, by default it will be true for "zscore", "nAnno" or "nOverlap"; otherwise false
`details`	logical to indicate whether the detailed information of terms is also viewed. By default, it sets to TRUE for the inclusion
`file`	a character specifying a file name written into. By default, it is 'Eoutput.txt'
`verbose`	logical to indicate whether the messages will be displayed in the screen. By default, it sets to true for display

Value

view(x) returns a data frame with following components:

- `term_id`: term ID
- `nAnno`: number in domains annotated by a term
- `nGroup`: number in domains from the input group
- `nOverlap`: number in overlaps
- `zscore`: enrichment z-score
- `pvalue`: p value
- `adjp`: adjusted p value

- `term_name`: term name
- `term_namespace`: term namespace; optional, it is only appended when "details" is true
- `term_distance`: term distance; optional, it is only appended when "details" is true
- `members`: members (represented as domain IDs) in overlaps; optional, it is only appended when "details" is true

write(x) also returns the same data frame as view(x), in addition to a specified local file.

See Also

`Eoutput-class`

`eTOL` *eukaryotic Tree Of Life (eTOL)*

Description

A 'phylo' object that contains information about eukaryotic part of species tree of life (eTOL). It is a rooted binary tree. Tips represent extant genomes. Since its reconstruction is guided under the NCBI taxonomy, each internal node is either mapped onto a unique taxonomic identifier or left empty (assumedly a hypothetical unknown ancestral genome).

Usage

```
data(eTOL)
```

Value

an object of class "phylo" with the following components:

- `Nnode`: the number of (internal) nodes
- `tip.label`: a vector giving the names of the tips (i.e., "left_id" to define the post-ordered binary tree structure)
- `node.label`: a vector giving the names of the internal nodes (i.e., "left_id" to define the post-ordered binary tree structure)
- `genome_info`: a matrix of all nodes (including tips and internal nodes) X 8, giving extant/ancestral genome information: "left_id" (unique and used as internal id), "right_id" (used in combination with "left_id" to define the post-ordered binary tree structure), "taxon_id" (NCBI taxonomy id, if matched), "genome" (2-letter genome identifiers used in SUPERFAMILY, if being extant), "name" (NCBI taxonomy name, if matched), "rank" (NCBI taxonomy rank, if matched), "branchlength" (branch length in relevance to the parent), and "common_name" (NCBI taxonomy common name, if matched and existed)
- `edge`: a two-column matrix of mode numeric where each row represents an edge of the tree; the nodes and the tips are symbolized with numbers; the tips are numbered 1, 2, ..., and the internal nodes are numbered after the tips. For each row, the first column gives the ancestor
- `edge.length`: a numeric vector giving the lengths of the branches given by 'edge'
- `root.length`: a numeric value giving the length of the branch at the root
- `connectivity`: a matrix of internal nodes X all nodes (including tips and internal nodes), with 1 for the presence of a ancestor-descenant path, and 0 otherwise

References

Fang et al. (2013) A daily-updated tree of (sequenced) life as a reference for genome research. *Scientific reports*, 3:2015.

Examples

```
data(eTOL)
eTOL
# list all components
names(eTOL)
# extract information about the first 5 genomes
eTOL$genome_info[1:5,]
# look at the dimension of connectivity
dim(eTOL$connectivity)
## Not run:
# visualise the connectivity matrix
Ntip <- length(eTOL$tip.label) # number of tips
Nnode <- eTOL$Nnode # number of internal nodes
data <- eTOL$connectivity
visHeatmapAdv(data, Rowv=FALSE,Colv=FALSE, zlim=c(0,1), colormap="gray-black", add.expr=abline(v=c(1,Ntip+1

## End(Not run)
```

Feature2GOBP.interpro *Annotations and scores between InterPro domains and Gene Ontology Biological Process (GOBP).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Biological Process terms and InterPro domains. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOBP.interpro <- dcRDataLoader(RData=Feature2GOBP.interpro)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOBP.interpro <- dcRDataLoader(RData=Feature2GOBP.interpro)
names(Feature2GOBP.interpro)
```

`Feature2GOBP.pfam` *Annotations and scores between Pfam domains/combinations and Gene Ontology Biological Process (GOBP).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Biological Process terms and Pfam domains/combinations. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOBP.pfam <- dcRDataLoader(RData=Feature2GOBP.pfam)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOBP.pfam <- dcRDataLoader(RData=Feature2GOBP.pfam)
names(Feature2GOBP.pfam)
```

`Feature2GOBP.sf` *Annotations and scores between SCOP domains/combinations (at the superfamily level; sf) and Gene Ontology Biological Process (GOBP).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Biological Process terms and SCOP domains/combinations (at the superfamily level; sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOBP.sf <- dcRDataLoader(RData=Feature2GOBP.sf)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOBP.sf <- dcRDataLoader(RData=Feature2GOBP.sf)
names(Feature2GOBP.sf)
```

`Feature2GOCC.interpro` *Annotations and scores between InterPro domains and Gene Ontology Cellular Component (GOCC).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Cellular Component terms and InterPro domains. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOCC.interpro <- dcRDataLoader(RData=Feature2GOCC.interpro)
```

Value

an object of class HIS. It has three components for "hscore", "ic" and "slim":

- hscore: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- ic: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- slim: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOCC.interpro <- dcRDataLoader(RData=Feature2GOCC.interpro)
names(Feature2GOCC.interpro)
```

Feature2GOCC.pfam — *Annotations and scores between Pfam domains/combinations and Gene Ontology Cellular Component (GOCC).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Cellular Component terms and Pfam domains/combinations. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOCC.pfam <- dcRDataLoader(RData=Feature2GOCC.pfam)
```

Value

an object of class HIS. It has three components for "hscore", "ic" and "slim":

- hscore: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- ic: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first

- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOCC.pfam <- dcRDataLoader(RData=Feature2GOCC.pfam)
names(Feature2GOCC.pfam)
```

`Feature2GOCC.sf` *Annotations and scores between SCOP domains/combinations (at the superfamily level; sf) and Gene Ontology Cellular Component (GOCC).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Cellular Component terms and SCOP domains/combinations (at the superfamily level; sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOCC.sf <- dcRDataLoader(RData=Feature2GOCC.sf)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOCC.sf <- dcRDataLoader(RData=Feature2GOCC.sf)
names(Feature2GOCC.sf)
```

Feature2GOMF.interpro *Annotations and scores between InterPro domains and Gene Ontology Molecular Function (GOMF).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Molecular Function terms and InterPro domains. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOMF.interpro <- dcRDataLoader(RData=Feature2GOMF.interpro)
```

Value

an object of class HIS. It has three components for "hscore", "ic" and "slim":

- hscore: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- ic: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- slim: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOMF.interpro <- dcRDataLoader(RData=Feature2GOMF.interpro)
names(Feature2GOMF.interpro)
```

`Feature2GOMF.pfam` *Annotations and scores between Pfam domains/combinations and Gene Ontology Molecular Function (GOMF).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Molecular Function terms and Pfam domains/combinations. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOMF.pfam <- dcRDataLoader(RData=Feature2GOMF.pfam)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOMF.pfam <- dcRDataLoader(RData=Feature2GOMF.pfam)
names(Feature2GOMF.pfam)
```

`Feature2GOMF.sf` *Annotations and scores between SCOP domains/combinations (at the superfamily level; sf) and Gene Ontology Molecular Function (GOMF).*

Description

An object of class "HIS" that contains annotations and scores between Gene Ontology Molecular Function terms and SCOP domains/combinations (at the superfamily level; sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2GOMF.sf <- dcRDataLoader(RData=Feature2GOMF.sf)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2GOMF.sf <- dcRDataLoader(RData=Feature2GOMF.sf)
names(Feature2GOMF.sf)
```

`Feature2HPPA.interpro` *Annotations and scores between InterPro domains and Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An object of class "HIS" that contains annotations and scores between Human Phenotype Phenotypic Abnormality terms and InterPro domains. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2HPPA.interpro <- dcRDataLoader(RData=Feature2HPPA.interpro)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first

- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2HPPA.interpro <- dcRDataLoader(RData=Feature2HPPA.interpro)
names(Feature2HPPA.interpro)
```

`Feature2HPPA.pfam` *Annotations and scores between Pfam domains/combinations and Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An object of class "HIS" that contains annotations and scores between Human Phenotype Phenotypic Abnormality terms and Pfam domains/combinations. This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2HPPA.pfam <- dcRDataLoader(RData=Feature2HPPA.pfam)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2HPPA.pfam <- dcRDataLoader(RData=Feature2HPPA.pfam)
names(Feature2HPPA.pfam)
```

`Feature2HPPA.sf` *Annotations and scores between SCOP domains/combinations (at the superfamily level; sf) and Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An object of class "HIS" that contains annotations and scores between Human Phenotype Phenotypic Abnormality terms and SCOP domains/combinations (at the superfamily level; sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
Feature2HPPA.sf <- dcRDataLoader(RData=Feature2HPPA.sf)
```

Value

an object of class `HIS`. It has three components for "hscore", "ic" and "slim":

- `hscore`: a list of domain-related features, each with a term-named vector containing hypergeometric score (hscore)
- `ic`: a term-named vector containing information content (IC). Terms are ordered first by IC and then by longest-path level, making sure that for terms with the same IC, parental terms always come first
- `slim`: a list of four slims, each with a term-named vector containing information content (IC). Slim '1' for very general terms, '2' for general terms, '3' for specific terms, '4' for very specific terms

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.
Fang H and Gough J. (2013) A domain-centric solution to functional genomics via dcGO Predictor. *BMC Bioinformatics*, 14(Suppl 3):S9.

Examples

```
# load data
Feature2HPPA.sf <- dcRDataLoader(RData=Feature2HPPA.sf)
names(Feature2HPPA.sf)
```

`InfoDataFrame-class` *Definition for S4 class InfoDataFrame*

Description

`InfoDataFrame` has two slots: data and dimLabels.

Value

Class InfoDataFrame

Slots

`data` A data.frame containing terms (rows) and measured variables (columns).

`dimLabels` A character descripting labels for rows and columns.

Creation

An object of this class can be created via: `new("InfoDataFrame",data, dimLabels)`

Methods

Class-specific methods:

- `dim()`: retrieve the dimension in the object
- `nrow()`: retrieve number of rows in the object
- `ncol()`: retrieve number of columns in the object
- `rowNames()`: retrieve names of rows in the object
- `colNames()`: retrieve names of columns in the object
- `dimLabels()`: retrieve the slot 'dimLabels', containing labels used for display of rows and columns in the object
- `Data()`: retrieve the slot 'data' in the object

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object
- `as(data.frame, "InfoDataFrame")`: convert a data.frame to an object of class InfoDataFrame
- `[i,j]`: get the subset of the same class

Access

Ways to access information on this class:

- `showClass("InfoDataFrame")`: show the class definition
- `showMethods(classes="InfoDataFrame")`: show the method definition upon this class
- `getSlots("InfoDataFrame")`: get the name and class of each slot in this class
- `slotNames("InfoDataFrame")`: get the name of each slot in this class
- `selectMethod(f, signature="InfoDataFrame")`: retrieve the definition code for the method 'f' defined in this class

See Also

InfoDataFrame-method

Examples

```
# generate data on domain information on
data <- data.frame(x=1:10, y=I(LETTERS[1:10]),
row.names=paste("Domain", 1:10, sep="_"))
dimLabels <- c("rowLabels", "colLabels")
# create an object of class InfoDataFrame
x <- new("InfoDataFrame", data=data, dimLabels=dimLabels)
x
# alternatively, using coerce methods
x <- as(data, "InfoDataFrame")
x
# look at various methods defined on class Anno
dimLabels(x)
dim(x)
nrow(x)
ncol(x)
rowNames(x)
colNames(x)
Data(x)
x[1:3,]
```

InfoDataFrame-method *Methods defined for S4 class InfoDataFrame*

Description

Methods defined for class InfoDataFrame.

Usage

```
dimLabels(x)

## S4 method for signature InfoDataFrame
dimLabels(x)

## S4 method for signature InfoDataFrame
dim(x)

## S4 method for signature InfoDataFrame
nrow(x)

## S4 method for signature InfoDataFrame
ncol(x)

## S4 method for signature InfoDataFrame
rowNames(x)

## S4 method for signature InfoDataFrame
```

```
colNames(x)

## S4 method for signature InfoDataFrame
Data(x)

## S4 method for signature InfoDataFrame
show(object)

## S4 method for signature InfoDataFrame,ANY,ANY,ANY
x[i, j, ..., drop = FALSE]
```

Arguments

x	an object of class InfoDataFrame
object	an object of class InfoDataFrame
i	an index
j	an index
...	additional parameters
drop	a logic for matrices and arrays. If TRUE the result is coerced to the lowest possible dimension. This only works for extracting elements, not for the replacement

See Also

InfoDataFrame-class

InterPro *InterPro domains (InterPro).*

Description

An object of class "InfoDataFrame" that contains information on InterPro domains (InterPro). This data is prepared based on ftp://anonymous@ftp.ebi.ac.uk/pub/databases/interpro/Current/entry.list.

Usage

```
data(InterPro)
```

Value

an object of class InfoDataFrame. It has slots for data and dimLabels:

- data: a data.frame containing information about 11638 annotatable domains (in rows), with 3 columns ("id" for InterPro ID, and "level" always equals "InterPro", "description" for InterPro description)
- dimLabels: a character describing labels for rows and columns in data

References

Hunter et al. (2012) InterPro in 2011: new developments in the family and domain prediction database. *Nucleic Acids Res*, 40(Database issue):D306-12.

See Also

InfoDataFrame-class

Examples

```
# load data
data(InterPro)
InterPro
# retrieve the dimension
dim(InterPro)
# retrieve names of columns
colNames(InterPro)
# retrieve the first 5 rows of data
Data(InterPro)[1:5,]
```

InterPro2GOBP *Annotations of InterPro domains by Gene Ontology Biological Process (GOBP).*

Description

An object of class "Anno" that contains associations between Gene Ontology Biological Process terms and InterPro domains. This data is prepared based on the InterPro database (see http://www.ebi.ac.uk/interpro/) and ftp://anonymous@ftp.ebi.ac.uk/pub/databases/interpro/Current/interpro2go.

Usage

```
data(InterPro2GOBP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for InterPro ID, and "level" always equals "InterPro", "description" for InterPro description

References

Hunter et al. (2012) Manual GO annotation of predictive protein signatures: the InterPro approach to GO curation. *Database (Oxford)*, 2012:bar068.

See Also

Anno-class

Examples

```
# load data
data(InterPro2GOBP)
InterPro2GOBP
# retrieve info on ontology terms
termData(InterPro2GOBP)
# retrieve info on InterPro domains
domainData(InterPro2GOBP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(InterPro2GOBP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

InterPro2GOCC *Annotations of InterPro domains by Gene Ontology Cellular Component (GOCC).*

Description

An object of class "Anno" that contains associations between Gene Ontology Cellular Component terms and InterPro domains. This data is prepared based on the InterPro database (see http://www.ebi.ac.uk/interpro/) and ftp://anonymous@ftp.ebi.ac.uk/pub/databases/interpro/Current/interpro2go.

Usage

```
data(InterPro2GOCC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for InterPro ID, and "level" always equals "InterPro", "description" for InterPro description

References

Hunter et al. (2012) Manual GO annotation of predictive protein signatures: the InterPro approach to GO curation. *Database (Oxford)*, 2012:bar068.

See Also

Anno-class

Examples

```
# load data
data(InterPro2GOCC)
InterPro2GOCC
# retrieve info on ontology terms
termData(InterPro2GOCC)
# retrieve info on InterPro domains
domainData(InterPro2GOCC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(InterPro2GOCC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

InterPro2GOMF *Annotations of InterPro domains by Gene Ontology Molecular Function (GOMF).*

Description

An object of class "Anno" that contains associations between Gene Ontology Molecular Function terms and InterPro domains. This data is prepared based on the InterPro database (see http://www.ebi.ac.uk/interpro/) and ftp://anonymous@ftp.ebi.ac.uk/pub/databases/interpro/Current/interpro2go.

Usage

```
data(InterPro2GOMF)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for InterPro ID, and "level" always equals "InterPro", "description" for InterPro description

References

Hunter et al. (2012) Manual GO annotation of predictive protein signatures: the InterPro approach to GO curation. *Database (Oxford)*, 2012:bar068.

See Also

Anno-class

Examples

```
# load data
data(InterPro2GOMF)
InterPro2GOMF
# retrieve info on ontology terms
termData(InterPro2GOMF)
# retrieve info on InterPro domains
domainData(InterPro2GOMF)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(InterPro2GOMF)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

Onto-class *Definition for S4 class Onto*

Description

`Onto` has 2 slots: nodeInfo and adjMatrix

Value

Class Onto

Slots

`nodeInfo` An object of S4 class `InfoDataFrame`, describing information on nodes/terms.

`adjMatrix` An object of S4 class `AdjData`, containing adjacency data matrix (for a direct graph), with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

Creation

An object of this class can be created via: `new("Onto", nodeInfo,adjMatrix)`

Methods

Class-specific methods:

- `dim()`: retrieve the dimension in the object
- `adjMatrix()`: retrieve the slot 'adjMatrix' in the object
- `nodeInfo()`: retrieve the slot 'nodeInfo' (as class InfoDataFrame) in the object
- `nInfo()`: retrieve nodeInfo (as data.frame) in the object
- `nodeNames()`: retrieve node/term names (ie, row names of nodeInfo) in the object
- `term_id()`: retrieve term id (ie, column 'term_id' of nodeInfo) in the object, if any
- `term_name()`: retrieve term id (ie, column 'term_name' of nodeInfo) in the object, if any
- `term_namespace()`: retrieve term id (ie, column 'term_namespace' of nodeInfo) in the object, if any
- `term_distance()`: retrieve term id (ie, column 'term_distance' of nodeInfo) in the object, if any

Standard generic methods:

- `str()`: compact display of the content in the object
- `show()`: abbreviated display of the object
- `as(matrix, "Onto")`: convert a matrix to an object of class Onto
- `as(dgCMatrix, "Onto")`: convert a dgCMatrix (a sparse matrix) to an object of class Onto
- `[i]`: get the subset of the same class

Access

Ways to access information on this class:

- `showClass("Onto")`: show the class definition
- `showMethods(classes="Onto")`: show the method definition upon this class
- `getSlots("Onto")`: get the name and class of each slot in this class
- `slotNames("Onto")`: get the name of each slot in this class
- `selectMethod(f, signature="Onto")`: retrieve the definition code for the method 'f' defined in this class

See Also

`Onto-method`

Examples

```
# create an object of class Onto, only given a matrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
as(adjM, "Onto")

# create an object of class Onto, given a matrix plus information on nodes
# 1) create nodeI: an object of class InfoDataFrame
data <- data.frame(term_id=paste("Term", 1:5, sep="_"),
term_name=I(LETTERS[1:5]), term_namespace=rep("Namespace",5),
term_distance=1:5, row.names=paste("Term", 1:5, sep="_"))
nodeI <- new("InfoDataFrame", data=data)
nodeI
# 2) create an object of class Onto
# VERY IMPORTANT: make sure having consistent names between nodeInfo and adjMatrix
adjM <- matrix(runif(25),nrow=5,ncol=5)
colnames(adjM) <- rownames(adjM) <- rowNames(nodeI)
x <- new("Onto", adjMatrix=adjM, nodeInfo=nodeI)
x
# 3) look at various methods defined on class Onto
dim(x)
adjMatrix(x)
nodeInfo(x)
nInfo(x)
nodeNames(x)
term_id(x)
term_namespace(x)
term_distance(x)
# 4) get the subset
x[1:2]
```

Onto-method *Methods defined for S4 class Onto*

Description

Methods defined for class Onto.

Usage

```
## S4 method for signature Onto
dim(x)

## S4 method for signature Onto
adjMatrix(x)

## S4 method for signature Onto
nodeInfo(x)

## S4 method for signature Onto
nInfo(object)

## S4 method for signature Onto
nodeNames(object)

## S4 method for signature Onto
term_id(object)

## S4 method for signature Onto
term_name(object)

## S4 method for signature Onto
term_namespace(object)

## S4 method for signature Onto
term_distance(object)

## S4 method for signature Onto
show(object)

## S4 method for signature Onto,ANY,ANY,ANY
x[i, j, ..., drop = FALSE]
```

Arguments

x	an object of class Onto
object	an object of class Onto
i	an index
j	an index
...	additional parameters
drop	a logic for matrices and arrays. If TRUE the result is coerced to the lowest possible dimension. This only works for extracting elements, not for the replacement

See Also

Onto-class

---

onto.DO *Disease Ontology (DO).*

Description

An R object that contains information on Gene Ontology Biological Process terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://sourceforge.net/p/diseaseontology/code/HEAD/tree/trunk/HumanDO.obo.

Usage

```
data(onto.DO)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Schriml et al. (2012) Disease Ontology: a backbone for disease semantic integration. *Nucleic Acids Res*, 40:D940-946.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.DO)
onto.DO
```

`onto.EC` *Enzyme Commission (EC).*

Description

An R object that contains information on Enzyme Commission terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). Enzyme Commission (EC) is a resource focused on enzyme nomenclature, which is a system of naming enzymes (protein catalysts) with Cross-references to UniProt sequences. It uses four-digit EC numbers to define the reaction catalysed. The first three digits are to define the reaction catalysed and the fourth for a unique identifier (serial number).

Usage

```
onto.EC <- dcRDataLoader(RData=onto.EC)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Fleischmann et al. (2004) IntEnz, the integrated relational enzyme database. *Nucleic Acids Res*, 32:D434-7.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
onto.EC <- dcRDataLoader(RData=onto.EC)
onto.EC
```

`onto.GOBP` *Gene Ontology Biological Process (GOBP).*

Description

An R object that contains information on Gene Ontology Biological Process terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology.1_2.obo. Only the edges with the relation (either 'is_a' or 'part_of') are retained.

Usage

```
data(onto.GOBP)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Ashburner et al. (2000) Gene ontology: tool for the unification of biology. *Nat Genet*, 25:25-29.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.GOBP)
onto.GOBP
```

`onto.GOCC` *Gene Ontology Cellular Component (GOCC).*

Description

An R object that contains information on Gene Ontology Cellular Component terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology.1_2.obo. Only the edges with the relation (either 'is_a' or 'part_of') are retained.

Usage

```
data(onto.GOCC)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Ashburner et al. (2000) Gene ontology: tool for the unification of biology. *Nat Genet*, 25:25-29.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.GOCC)
onto.GOCC
```

`onto.GOMF` *Gene Ontology Molecular Function (GOMF).*

Description

An R object that contains information on Gene Ontology Molecular Function terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see `http://igraph.org/r/doc/aaa-igraph-package.html`). This data is prepared based on `http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology.1_2.obo`. Only the edges with the relation (either 'is_a' or 'part_of') are retained.

Usage

```
data(onto.GOMF)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Ashburner et al. (2000) Gene ontology: tool for the unification of biology. *Nat Genet*, 25:25-29.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.GOMF)
onto.GOMF
```

onto.HPMI *Human Phenotype Mode of Inheritance (HPMI).*

Description

An R object that contains information on Human Phenotype Mode of Inheritance terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://compbio.charite.de/svn/hpo/trunk/src/ontology/human-phenotype-ontology.obo.

Usage

```
data(onto.HPMI)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Robinson et al. (2012) The Human Phenotype Ontology: a tool for annotating and analyzing human hereditary disease. *Am J Hum Genet*, 83:610-615.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.HPMI)
onto.HPMI
```

onto.HPON *Human Phenotype ONset and clinical course (HPON).*

Description

An R object that contains information on Human Phenotype ONset and clinical course terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://compbio.charite.de/svn/hpo/trunk/src/ontology/human-phenotype-ontology.obo.

Usage

```
data(onto.HPON)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Robinson et al. (2012) The Human Phenotype Ontology: a tool for annotating and analyzing human hereditary disease. *Am J Hum Genet*, 83:610-615.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.HPON)
onto.HPON
```

`onto.HPPA` *Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An R object that contains information on Human Phenotype Phenotypic Abnormality terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see `http://igraph.org/r/doc/aaa-igraph-package.html`). This data is prepared based on `http://compbio.charite.de/svn/hpo/trunk/src/ontology/human-phenotype-ontology.obo`.

Usage

```
data(onto.HPPA)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Robinson et al. (2012) The Human Phenotype Ontology: a tool for annotating and analyzing human hereditary disease. *Am J Hum Genet*, 83:610-615.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.HPPA)
onto.HPPA
```

`onto.KW` *UniProtKB KeyWords (KW).*

Description

An R object that contains information on UniProtKB KeyWords terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). UniProtKB KeyWords (KW) controlled vocabulary provides a summary of the entry content and is used to index UniProtKB/Swiss-Prot entries based on 10 categories (the category "Technical term" being excluded here).

Usage

```
onto.KW <- dcRDataLoader(RData=onto.KW)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Bairoch et al. (2005) The Universal Protein Resource (UniProt). *Nucleic Acids Res*, 33:D154-9.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
onto.KW <- dcRDataLoader(RData=onto.KW)
onto.KW
```

onto.MP *Mammalian Phenotype (MP).*

Description

An R object that contains information on Mammalian Phenotype terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). This data is prepared based on http://sourceforge.net/p/diseaseontology/code/HEAD/tree/trunk/HumanMP.obo.

Usage

```
data(onto.MP)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Smith et al. (2009) The Mammalian Phenotype Ontology: enabling robust annotation and comparative analysis. *Wiley Interdiscip Rev Syst Biol Med*, 1:390-399.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
data(onto.MP)
onto.MP
```

onto.UP *UniProtKB UniPathway (UP).*

Description

An R object that contains information on UniProtKB UniPathway terms. These terms are organised as a direct acyclic graph (DAG), which is further stored as an object of the class 'igraph' (see http://igraph.org/r/doc/aaa-igraph-package.html). UniProtKB UniPathway (UP) is a fully manually curated resource for the representation and annotation of metabolic pathways, being used as a controlled vocabulary for pathway annotation in UniProtKB.

Usage

```
onto.UP <- dcRDataLoader(RData=onto.UP)
```

Value

an object of S4 class Onto. It has slots for "nodeInfo" and "adjMatrix"

- `nodeInfo`: an object of S4 class InfoDataFrame, describing information on nodes/terms including: "term_id" (i.e. Term ID), "term_name" (i.e. Term Name), "term_namespace" (i.e. Term Namespace), and "term_distance" (i.e. Term Distance: the distance to the root; always 0 for the root itself)
- `adjMatrix`: an object of S4 class AdjData, containing adjacency data matrix, with rows for parent (arrow-outbound) and columns for children (arrow-inbound)

References

Morgat et al. (2006) UniPathway: a resource for the exploration and annotation of metabolic pathways. *Nucleic Acids Res*, 40:D761-9.
Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

Examples

```
onto.UP <- dcRDataLoader(RData=onto.UP)
onto.UP
```

`Pfam` *Pfam domains (Pfam).*

Description

An object of class "InfoDataFrame" that contains information on Pfam domains (Pfam). This data is prepared based on `ftp://ftp.sanger.ac.uk/pub/databases/Pfam/current_release/database_files/pfamA.txt.gz`.

Usage

```
data(Pfam)
```

Value

an object of class `InfoDataFrame`. It has slots for data and dimLabels:

- `data`: a data.frame containing information about 14831 domains (in rows), with 3 columns ("id" for Pfam accession ID, and "level" always equals "Pfam", "description" for Pfam description)
- `dimLabels`: a character describing labels for rows and columns in data

References

Finn et al. (2014) The Pfam protein families database. *Nucleic Acids Res*, 42(Database issue):D222-D230.

See Also

`InfoDataFrame-class`

Examples

```
# load data
data(Pfam)
Pfam
# retrieve the dimension
dim(Pfam)
# retrieve names of columns
colNames(Pfam)
# retrieve the first 5 rows of data
Data(Pfam)[1:5,]
```

Pfam2GOBP — *Annotations of Pfam domains by Gene Ontology Biological Process (GOBP).*

Description

An object of class "Anno" that contains associations between Gene Ontology Biological Process terms and Pfam domains. This data is prepared based on the Pfam database (see http://pfam.xfam.org) and ftp://ftp.geneontology.org/pub/go/external2go/pfam2go.

Usage

```
data(Pfam2GOBP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for Pfam accession ID, and "level" always equals "Pfam", "description" for Pfam description

References

Finn et al. (2014) The Pfam protein families database. *Nucleic Acids Res*, 42(Database issue):D222-D230.

See Also

Anno-class

Examples

```
# load data
data(Pfam2GOBP)
Pfam2GOBP
# retrieve info on ontology terms
termData(Pfam2GOBP)
# retrieve info on Pfam domains
domainData(Pfam2GOBP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Pfam2GOBP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

`Pfam2GOCC` *Annotations of Pfam domains by Gene Ontology Cellular Component (GOCC).*

Description

An object of class "Anno" that contains associations between Gene Ontology Cellular Component terms and Pfam domains. This data is prepared based on the Pfam database (see http://pfam.xfam.org) and ftp://ftp.geneontology.org/pub/go/external2go/pfam2go.

Usage

```
data(Pfam2GOCC)
```

Value

an object of class `Anno`. It has slots for "annoData", "termData" and "domainData":

- `annoData`: a sparse matrix of domains X terms
- `termData`: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- `domainData`: variables describing domains (i.e. rows in annoData), including: "id" for Pfam accession ID, and "level" always equals "Pfam", "description" for Pfam description

References

Finn et al. (2014) The Pfam protein families database. *Nucleic Acids Res*, 42(Database issue):D222-D230.

See Also

`Anno-class`

Examples

```
# load data
data(Pfam2GOCC)
Pfam2GOCC
# retrieve info on ontology terms
termData(Pfam2GOCC)
# retrieve info on Pfam domains
domainData(Pfam2GOCC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Pfam2GOCC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

Pfam2GOMF *Annotations of Pfam domains by Gene Ontology Molecular Function (GOMF).*

Description

An object of class "Anno" that contains associations between Gene Ontology Molecular Function terms and Pfam domains. This data is prepared based on the Pfam database (see http://pfam.xfam.org) and ftp://ftp.geneontology.org/pub/go/external2go/pfam2go.

Usage

```
data(Pfam2GOMF)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for Pfam accession ID, and "level" always equals "Pfam", "description" for Pfam description

References

Finn et al. (2014) The Pfam protein families database. *Nucleic Acids Res*, 42(Database issue):D222-D230.

See Also

Anno-class

Examples

```
# load data
data(Pfam2GOMF)
Pfam2GOMF
# retrieve info on ontology terms
termData(Pfam2GOMF)
# retrieve info on Pfam domains
domainData(Pfam2GOMF)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Pfam2GOMF)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

Rfam *RNA families (Rfam).*

Description

An object of class "InfoDataFrame" that contains information on RNA families (Rfam). This data is prepared based on ftp://anonymous@ftp.sanger.ac.uk/pub/databases/Rfam/11.0/database_files/rfam.txt.gz.

Usage

```
data(Rfam)
```

Value

an object of class InfoDataFrame. It has slots for data and dimLabels:

- `data`: a data.frame containing information about 2208 RNA families (in rows), with 3 columns ("id" for Rfam accession ID, and "level" always equals "Rfam", "description" for Rfam description)
- `dimLabels`: a character describing labels for rows and columns in data

References

Gardner et al. (2011) Rfam: Wikipedia, clans and the "decimal" release. *Nucleic Acids Res*, 39(Database issue):D141-D145.

See Also

InfoDataFrame-class

Examples

```
# load data
data(Rfam)
Rfam
# retrieve the dimension
dim(Rfam)
# retrieve names of columns
```

```
colNames(Rfam)
# retrieve the first 5 rows of data
Data(Rfam)[1:5,]
```

Rfam2GOBP *Annotations of Rfam RNA families by Gene Ontology Biological Process (GOBP).*

Description

An object of class "Anno" that contains associations between Gene Ontology Biological Process terms and Rfam RNA families. This data is prepared based on the Rfam database (see http://rfam.xfam.org) and http://geneontology.org/external2go/rfam2go.

Usage

```
data(Rfam2GOBP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of RNAs X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing RNAs (i.e. rows in annoData), including: "id" for Rfam accession ID, and "level" always equals "Rfam", "description" for Rfam description

References

Gardner et al. (2011) Rfam: Wikipedia, clans and the "decimal" release. *Nucleic Acids Res*, 39(Database issue):D141-D145.

See Also

Anno-class

Examples

```
# load data
data(Rfam2GOBP)
Rfam2GOBP
# retrieve info on ontology terms
termData(Rfam2GOBP)
# retrieve info on Rfam RNAs
domainData(Rfam2GOBP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Rfam2GOBP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

Rfam2GOCC *Annotations of Rfam RNA families by Gene Ontology Cellular Component (GOCC).*

Description

An object of class "Anno" that contains associations between Gene Ontology Cellular Component terms and Rfam RNA families. This data is prepared based on the Rfam database (see http://rfam.xfam.org) and http://geneontology.org/external2go/rfam2go.

Usage

```
data(Rfam2GOCC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of RNAs X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing RNAs (i.e. rows in annoData), including: "id" for Rfam accession ID, and "level" always equals "Rfam", "description" for Rfam description

References

Gardner et al. (2011) Rfam: Wikipedia, clans and the "decimal" release. *Nucleic Acids Res*, 39(Database issue):D141-D145.

See Also

Anno-class

Examples

```
# load data
data(Rfam2GOCC)
Rfam2GOCC
# retrieve info on ontology terms
termData(Rfam2GOCC)
# retrieve info on Rfam RNAs
domainData(Rfam2GOCC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Rfam2GOCC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

Rfam2GOMF *Annotations of Rfam RNA families by Gene Ontology Molecular Function (GOMF).*

Description

An object of class "Anno" that contains associations between Gene Ontology Molecular Function terms and Rfam RNA families. This data is prepared based on the Rfam database (see http://rfam.xfam.org) and http://geneontology.org/external2go/rfam2go.

Usage

```
data(Rfam2GOMF)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of RNAs X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing RNAs (i.e. rows in annoData), including: "id" for Rfam accession ID, and "level" always equals "Rfam", "description" for Rfam description

References

Gardner et al. (2011) Rfam: Wikipedia, clans and the "decimal" release. *Nucleic Acids Res*, 39(Database issue):D141-D145.

See Also

Anno-class

Examples

```
# load data
data(Rfam2GOMF)
Rfam2GOMF
# retrieve info on ontology terms
termData(Rfam2GOMF)
# retrieve info on Rfam RNAs
domainData(Rfam2GOMF)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(Rfam2GOMF)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa *SCOP domain families (fa).*

Description

An object of class "InfoDataFrame" that contains information on SCOP domain families (fa).

Usage

```
data(SCOP.fa)
```

Value

an object of class InfoDataFrame. It has slots for data and dimLabels:

- data: a data.frame containing information about 2223 domains (in rows), with 3 columns ("id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description)
- dimLabels: a character describing labels for rows and columns in data

References

Morais et al. (2011) SUPERFAMILY 1.75 including a domain-centric gene ontology method. *Nucleic Acids Res*, 39(Database issue):D427-34.
Andreeva et al. (2008) Data growth and its impact on the SCOP database: new developments. *Nucleic Acids Res*, 36(Database issue):D419-425

See Also

InfoDataFrame-class

Examples

```
# load data
data(SCOP.fa)
SCOP.fa
# retrieve the dimension
dim(SCOP.fa)
# retrieve names of columns
colNames(SCOP.fa)
# retrieve the first 5 rows of data
Data(SCOP.fa)[1:5,]
```

SCOP.fa2DO *Annotations of SCOP domain families (fa) by Disease Ontology (DO).*

Description

An object of class "Anno" that contains associations between Disease Ontology terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2DO)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2DO)
SCOP.fa2DO
# retrieve info on ontology terms
termData(SCOP.fa2DO)
# retrieve info on SCOP domains
domainData(SCOP.fa2DO)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2DO)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2EC *Annotations of SCOP domain families (fa) by Enzyme Commission (EC).*

Description

An object of class "Anno" that contains associations between Enzyme Commission terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.fa2EC <- dcRDataLoader(RData=SCOP.fa2EC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- `annoData`: a sparse matrix of domains X terms
- `termData`: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- `domainData`: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.fa2EC <- dcRDataLoader(RData=SCOP.fa2EC)
SCOP.fa2EC
# retrieve info on ontology terms
termData(SCOP.fa2EC)
# retrieve info on SCOP domains
domainData(SCOP.fa2EC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2EC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2GOBP *Annotations of SCOP domain families (fa) by Gene Ontology Biological Process (GOBP).*

Description

An object of class "Anno" that contains associations between Gene Ontology Biological Process terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2GOBP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2GOBP)
SCOP.fa2GOBP
# retrieve info on ontology terms
termData(SCOP.fa2GOBP)
# retrieve info on SCOP domains
domainData(SCOP.fa2GOBP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2GOBP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2GOCC *Annotations of SCOP domain families (fa) by Gene Ontology Cellular Component (GOCC).*

Description

An object of class "Anno" that contains associations between Gene Ontology Cellular Component terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2GOCC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2GOCC)
SCOP.fa2GOCC
# retrieve info on ontology terms
termData(SCOP.fa2GOCC)
# retrieve info on SCOP domains
domainData(SCOP.fa2GOCC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2GOCC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2GOMF *Annotations of SCOP domain families (fa) by Gene Ontology Molecular Function (GOMF).*

Description

An object of class "Anno" that contains associations between Gene Ontology Molecular Function terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2GOMF)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2GOMF)
SCOP.fa2GOMF
# retrieve info on ontology terms
termData(SCOP.fa2GOMF)
# retrieve info on SCOP domains
domainData(SCOP.fa2GOMF)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2GOMF)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2HPMI *Annotations of SCOP domain families (fa) by Human Phenotype Mode of Inheritance (HPMI).*

Description

An object of class "Anno" that contains associations between HPMI terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2HPMI)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2HPMI)
SCOP.fa2HPMI
# retrieve info on ontology terms
termData(SCOP.fa2HPMI)
# retrieve info on SCOP domains
domainData(SCOP.fa2HPMI)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2HPMI)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2HPON *Annotations of SCOP domain families (fa) by Human Phenotype ONset and clinical course (HPON).*

Description

An object of class "Anno" that contains associations between HPON terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2HPON)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2HPON)
SCOP.fa2HPON
# retrieve info on ontology terms
termData(SCOP.fa2HPON)
# retrieve info on SCOP domains
domainData(SCOP.fa2HPON)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2HPON)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2HPPA *Annotations of SCOP domain families (fa) by Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An object of class "Anno" that contains associations between HPPA terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2HPPA)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2HPPA)
SCOP.fa2HPPA
# retrieve info on ontology terms
termData(SCOP.fa2HPPA)
# retrieve info on SCOP domains
domainData(SCOP.fa2HPPA)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2HPPA)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2KW *Annotations of SCOP domain families (fa) by UniProtKB KeyWords (KW).*

Description

An object of class "Anno" that contains associations between UniProtKB KeyWords terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.fa2KW <- dcRDataLoader(RData=SCOP.fa2KW)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.fa2KW <- dcRDataLoader(RData=SCOP.fa2KW)
SCOP.fa2KW
# retrieve info on ontology terms
termData(SCOP.fa2KW)
# retrieve info on SCOP domains
domainData(SCOP.fa2KW)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2KW)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2MP *Annotations of SCOP domain families (fa) by Mammalian Phenotype (MP).*

Description

An object of class "Anno" that contains associations between Mammalian Phenotype terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.fa2MP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.fa2MP)
SCOP.fa2MP
# retrieve info on ontology terms
termData(SCOP.fa2MP)
# retrieve info on SCOP domains
domainData(SCOP.fa2MP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2MP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.fa2UP *Annotations of SCOP domain families (fa) by UniProtKB UniPathway (UP).*

Description

An object of class "Anno" that contains associations between UniProtKB UniPathway terms and SCOP domain families (fa). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.fa2UP <- dcRDataLoader(RData=SCOP.fa2UP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.fa2UP <- dcRDataLoader(RData=SCOP.fa2UP)
SCOP.fa2UP
# retrieve info on ontology terms
termData(SCOP.fa2UP)
# retrieve info on SCOP domains
domainData(SCOP.fa2UP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.fa2UP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf *SCOP domain superfamilies (sf).*

Description

An object of class "InfoDataFrame" that contains information on SCOP domain superfamilies (sf).

Usage

```
data(SCOP.sf)
```

Value

an object of class InfoDataFrame. It has slots for data and dimLabels:

- data: a data.frame containing information about 2223 domains (in rows), with 3 columns ("id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description)
- dimLabels: a character describing labels for rows and columns in data

References

Morais et al. (2011) SUPERFAMILY 1.75 including a domain-centric gene ontology method. *Nucleic Acids Res*, 39(Database issue):D427-34.
Andreeva et al. (2008) Data growth and its impact on the SCOP database: new developments. *Nucleic Acids Res*, 36(Database issue):D419-425

See Also

InfoDataFrame-class

Examples

```
# load data
data(SCOP.sf)
SCOP.sf
# retrieve the dimension
dim(SCOP.sf)
# retrieve names of columns
colNames(SCOP.sf)
# retrieve the first 5 rows of data
Data(SCOP.sf)[1:5,]
```

SCOP.sf2DO — *Annotations of SCOP domain superfamilies (sf) by Disease Ontology (DO).*

Description

An object of class "Anno" that contains associations between Disease Ontology terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2DO)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2DO)
SCOP.sf2DO
# retrieve info on ontology terms
termData(SCOP.sf2DO)
# retrieve info on SCOP domains
domainData(SCOP.sf2DO)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2DO)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2EC *Annotations of SCOP domain superfamilies (sf) by Enzyme Commission (EC).*

Description

An object of class "Anno" that contains associations between Enzyme Commission terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.sf2EC <- dcRDataLoader(RData=SCOP.sf2EC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.sf2EC <- dcRDataLoader(RData=SCOP.sf2EC)
SCOP.sf2EC
# retrieve info on ontology terms
termData(SCOP.sf2EC)
# retrieve info on SCOP domains
domainData(SCOP.sf2EC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2EC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2GOBP *Annotations of SCOP domain superfamilies (sf) by Gene Ontology Biological Process (GOBP).*

Description

An object of class "Anno" that contains associations between Gene Ontology Biological Process terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2GOBP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2GOBP)
SCOP.sf2GOBP
# retrieve info on ontology terms
termData(SCOP.sf2GOBP)
# retrieve info on SCOP domains
domainData(SCOP.sf2GOBP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2GOBP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2GOCC *Annotations of SCOP domain superfamilies (sf) by Gene Ontology Cellular Component (GOCC).*

Description

An object of class "Anno" that contains associations between Gene Ontology Cellular Component terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2GOCC)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2GOCC)
SCOP.sf2GOCC
# retrieve info on ontology terms
termData(SCOP.sf2GOCC)
# retrieve info on SCOP domains
domainData(SCOP.sf2GOCC)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2GOCC)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2GOMF *Annotations of SCOP domain superfamilies (sf) by Gene Ontology Molecular Function (GOMF).*

Description

An object of class "Anno" that contains associations between Gene Ontology Molecular Function terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2GOMF)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2GOMF)
SCOP.sf2GOMF
# retrieve info on ontology terms
termData(SCOP.sf2GOMF)
# retrieve info on SCOP domains
domainData(SCOP.sf2GOMF)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2GOMF)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2HPMI *Annotations of SCOP domain superfamilies (sf) by Human Phenotype Mode of Inheritance (HPMI).*

Description

An object of class "Anno" that contains associations between HPMI terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2HPMI)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2HPMI)
SCOP.sf2HPMI
# retrieve info on ontology terms
termData(SCOP.sf2HPMI)
# retrieve info on SCOP domains
domainData(SCOP.sf2HPMI)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2HPMI)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2HPON *Annotations of SCOP domain superfamilies (sf) by Human Phenotype ONset and clinical course (HPON).*

Description

An object of class "Anno" that contains associations between HPON terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2HPON)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2HPON)
SCOP.sf2HPON
# retrieve info on ontology terms
termData(SCOP.sf2HPON)
# retrieve info on SCOP domains
domainData(SCOP.sf2HPON)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2HPON)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2HPPA *Annotations of SCOP domain superfamilies (sf) by Human Phenotype Phenotypic Abnormality (HPPA).*

Description

An object of class "Anno" that contains associations between HPPA terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2HPPA)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2HPPA)
SCOP.sf2HPPA
# retrieve info on ontology terms
termData(SCOP.sf2HPPA)
# retrieve info on SCOP domains
domainData(SCOP.sf2HPPA)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2HPPA)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2KW *Annotations of SCOP domain superfamilies (sf) by UniProtKB KeyWords (KW).*

Description

An object of class "Anno" that contains associations between UniProtKB KeyWords terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.sf2KW <- dcRDataLoader(RData=SCOP.sf2KW)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.sf2KW <- dcRDataLoader(RData=SCOP.sf2KW)
SCOP.sf2KW
# retrieve info on ontology terms
termData(SCOP.sf2KW)
# retrieve info on SCOP domains
domainData(SCOP.sf2KW)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2KW)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2MP — *Annotations of SCOP domain superfamilies (sf) by Mammalian Phenotype (MP).*

Description

An object of class "Anno" that contains associations between Mammalian Phenotype terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
data(SCOP.sf2MP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- `annoData`: a sparse matrix of domains X terms
- `termData`: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- `domainData`: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
data(SCOP.sf2MP)
SCOP.sf2MP
# retrieve info on ontology terms
termData(SCOP.sf2MP)
# retrieve info on SCOP domains
domainData(SCOP.sf2MP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2MP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

SCOP.sf2UP — *Annotations of SCOP domain superfamilies (sf) by UniProtKB UniPathway (UP).*

Description

An object of class "Anno" that contains associations between UniProtKB UniPathway terms and SCOP domain superfamilies (sf). This data is prepared based on the dcGO database (see http://supfam.org/SUPERFAMILY/dcGO/).

Usage

```
SCOP.sf2UP <- dcRDataLoader(RData=SCOP.sf2UP)
```

Value

an object of class Anno. It has slots for "annoData", "termData" and "domainData":

- annoData: a sparse matrix of domains X terms
- termData: variables describing ontology terms (i.e. columns in annoData), including: "ID" (i.e. term ID), "Name" (i.e. term Names), "Namespace" (i.e. term Namespace), and "Distance" (i.e. term Distance to the ontology root)
- domainData: variables describing domains (i.e. rows in annoData), including: "id" for SCOP sunid, and "level" for SCOP level, "description" for SCOP description

References

Fang H and Gough J. (2013) dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more. *Nucleic Acids Res*, 41(Database issue):D536-44.

See Also

Anno-class

Examples

```
# load data
SCOP.sf2UP <- dcRDataLoader(RData=SCOP.sf2UP)
SCOP.sf2UP
# retrieve info on ontology terms
termData(SCOP.sf2UP)
# retrieve info on SCOP domains
domainData(SCOP.sf2UP)
# retrieve the first 5 rows and columns of annotation data
x <- annoData(SCOP.sf2UP)[1:5,1:5]
x
# convert the above retrieval to the full matrix
as.matrix(x)
```

`visEnrichment` *Function to visualise enrichment analysis outputs in the context of the ontology hierarchy*

Description

`visEnrichment` is supposed to visualise enrichment analysis outputs (represented as an 'Eoutput' object) in the context of the ontology hierarchy (direct acyclic graph; DAG). Only part of DAG induced by those nodes/terms specified in query nodes (and the mode defining the paths to the root of DAG) will be visualised. Nodes in query are framed in black (by default), and all nodes (in query plus induced) will be color-coded according to a given data.type ('zscore'; otherwise taking the form of 10-based negative logarithm for 'adjp' or 'pvalue'). If no nodes in query, the top 5 significant terms (in terms of adjusted p-value) will be used for visualisation

Usage

```
visEnrichment(e, nodes_query = NULL, num_top_nodes = 5,
path.mode = c("all_shortest_paths", "shortest_paths", "all_paths"),
data.type = c("adjp", "pvalue", "zscore"), height = 7, width = 7,
margin = rep(0.1, 4), colormap = c("yr", "bwr", "jet", "gbr", "wyr",
"br",
"rainbow", "wb", "lightyellow-orange"), ncolors = 40, zlim = NULL,
colorbar = T, colorbar.fraction = 0.1, newpage = T,
layout.orientation = c("left_right", "top_bottom", "bottom_top",
"right_left"), node.info = c("both", "none", "term_id", "term_name",
"full_term_name"), graph.node.attrs = NULL, graph.edge.attrs = NULL,
node.attrs = NULL)
```

Arguments

`e`	an object of S4 class `Eoutput`
`nodes_query`	a verctor containing a list of nodes/terms in query. These nodes are used to produce a subgraph of the ontology DAG induced by them. If NULL, the top significant terms (in terms of p-value) will be determined by the next 'num_top_nodes'
`num_top_nodes`	a numeric value specifying the number of the top significant terms (in terms of p-value) will be used. This parameter does not work if the previous 'nodes_query' has been specified
`path.mode`	the mode of paths induced by nodes in query. 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)
`data.type`	a character telling which data type for nodes in query is used to color-code nodes. It can be one of 'adjp' for adjusted p-values (by default), 'pvalue' for p-values and 'zscore' for z-scores. When 'adjp' or 'pvalue' is used, 10-based negative logarithm is taken. For the style of how to color-code, please see the next arguments: colormap, ncolors, zlim and colorbar
`height`	a numeric value specifying the height of device
`width`	a numeric value specifying the width of device
`margin`	margins as units of length 4 or 1

`colormap` short name for the colormap. It can be one of "yr" (yellow-red colormap; by default), "jet" (jet colormap), "bwr" (blue-white-red colormap), "gbr" (green-black-red colormap), "wyr" (white-yellow-red colormap), "br" (black-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. "lightyellow-orange" (by default), "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/data 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 data is null, it always sets to false

`colorbar.fraction` the relative fraction of colorbar block against the device size

`newpage` logical to indicate whether to open a new page. By default, it sets to true for opening a new page

`layout.orientation` the orientation of the DAG layout. It can be one of "left_right" for the left-right layout (viewed from the DAG root point; by default), "top_bottom" for the top-bottom layout, "bottom_top" for the bottom-top layout, and "right_left" for the right-left layout

`node.info` tells the ontology term information used to label nodes. It can be one of "both" for using both of Term ID and Name (the first 15 characters; by default), "none" for no node labeling, "term_id" for using Term ID, "term_name" for using Term Name (the first 15 characters), and "full_term_name" for using the full Term Name

`graph.node.attrs` a list of global node attributes. These node attributes will be changed globally. See 'Note' below for details on the attributes

`graph.edge.attrs` a list of global edge attributes. These edge attributes will be changed globally. See 'Note' below for details on the attributes

`node.attrs` a list of local edge attributes. These node attributes will be changed locally; as such, for each attribute, the input value must be a named vector (i.e. using Term ID as names). See 'Note' below for details on the attributes

Value

An object of class 'Ragraph'

Note

A list of global node attributes used in "graph.node.attrs":

- "shape": the shape of the node: "circle", "rectangle", "rect", "box" and "ellipse"
- "fixedsize": the logical to use only width and height attributes. By default, it sets to true for not expanding for the width of the label
- "fillcolor": the background color of the node

- "color": the color for the node, corresponding to the outside edge of the node
- "fontcolor": the color for the node text/labelings
- "fontsize": the font size for the node text/labelings
- "height": the height (in inches) of the node: 0.5 by default
- "width": the width (in inches) of the node: 0.75 by default
- "style": the line style for the node: "solid", "dashed", "dotted", "invis" and "bold"

A list of global edge attributes used in "graph.edge.attrs":

- "color": the color of the edge: gray by default
- "weight": the weight of the edge: 1 by default
- "style": the line style for the edge: "solid", "dashed", "dotted", "invis" and "bold"

A list of local node attributes used in "node.attrs" (only those named Term IDs will be changed locally!):

- "label": a named vector specifying the node text/labelings
- "shape": a named vector specifying the shape of the node: "circle", "rectangle", "rect", "box" and "ellipse"
- "fixedsize": a named vector specifying whether it sets to true for not expanding for the width of the label
- "fillcolor": a named vector specifying the background color of the node
- "color": a named vector specifying the color for the node, corresponding to the outside edge of the node
- "fontcolor": a named vector specifying the color for the node text/labelings
- "fontsize": a named vector specifying the font size for the node text/labelings
- "height": a named vector specifying the height (in inches) of the node: 0.5 by default
- "width": a named vector specifying the width (in inches) of the node: 0.75 by default
- "style": a named vector specifying the line style for the node: "solid", "dashed", "dotted", "invis" and "bold"

See Also

dcEnrichment, dcRDataLoader, dcConverter

Examples

```
# 1) load SCOP.sf (as InfoDataFrame object)
SCOP.sf <- dcRDataLoader(SCOP.sf)
# randomly select 20 domains
data <- sample(rowNames(SCOP.sf), 20)

# 2) perform enrichment analysis, producing an object of S4 class Eoutput
eoutput <- dcEnrichment(data, domain="SCOP.sf", ontology="GOMF")
eoutput

# 3) visualise the top 10 significant terms
# color-coded according to 10-based negative logarithm of p-values
visEnrichment(eoutput)
# color-coded according to zscore
visEnrichment(eoutput, data.type=zscore)
```

```
# 4) visualise the top 5 significant terms in the ontology hierarchy
nodes_query <- names(sort(adjp(eoutput))[1:5])
visEnrichment(eoutput, nodes_query=nodes_query)
# change the frame color: highlight (framed in blue) nodes/terms in query
nodes.highlight <- rep("blue", length(nodes_query))
names(nodes.highlight) <- nodes_query
visEnrichment(eoutput, nodes_query=nodes_query,
node.attrs=list(color=nodes.highlight))
```

Index