

Package ‘CIMICE’

August 1, 2021

Type Package

Title CIMICE-R: (Markov) Chain Method to Inferr Cancer Evolution

Version 1.0.0

Description CIMICE is a tool in the field of tumor phylogenetics and its goal is to build a Markov Chain (called Cancer Progression Markov Chain, CPMC) in order to model tumor subtypes evolution.

The input of CIMICE is a Mutational Matrix, so a boolean matrix representing altered genes in a collection of samples. These samples are assumed to be obtained with single-cell DNA analysis techniques and the tool is specifically written to use the peculiarities of this data for the CMPC construction.

License Artistic-2.0

Encoding UTF-8

Imports dplyr, ggplot2, glue, tidyr, igraph, networkD3, visNetwork, ggcrrplot, purrr, ggraph, stats, utils, relations, maftools, assertthat, Matrix

RoxygenNote 7.1.1

VignetteBuilder knitr

Suggests BiocStyle, knitr, rmarkdown, testthat, webshot

biocViews Software, BiologicalQuestion, NetworkInference, ResearchField, Phylogenetics, StatisticalMethod, GraphAndNetwork, Technology, SingleCell

BugReports <https://github.com/redsnic/CIMICE/issues>

URL <https://github.com/redsnic/CIMICE>

BiocType Software

git_url <https://git.bioconductor.org/packages/CIMICE>

git_branch RELEASE_3_13

git_last_commit 4e2d4f7

git_last_commit_date 2021-05-19

Date/Publication 2021-08-01

Author Nicolò Rossi [aut, cre] (Lab. of Computational Biology and Bioinformatics, Department of Mathematics, Computer Science and Physics, University of Udine,
<<https://orcid.org/0000-0002-6353-7396>>)

Maintainer Nicolò Rossi <olocin.issor@gmail.com>

R topics documented:

annotate_mutational_matrix	3
binary_radix_sort	4
build_subset_graph	4
build_topology_subset	5
chunk_reader	6
CIMICE	7
compact_dataset	7
computeDWNW	8
computeDWNW_aux	9
computeUPW	9
computeUPW_aux	10
compute_weights_default	11
corrplot_from_mutational_matrix	11
corrplot_genes	12
corrplot_samples	13
dataset_preprocessing	13
dataset_preprocessing_population	14
draw_ggraph	15
draw_networkD3	15
draw_visNetwork	16
example_dataset	17
example_dataset_withFreqs	17
fix_clonal_genotype	18
gene_mutations_hist	19
get_no_of_children	19
graph_non_transitive_subset_topology	20
make_dataset	21
normalizeDWNW	21
normalizeUPW	22
prepare_labels	23
quick_run	24
read	25
read_CAPRI	25
read_CAPRIpop	26
read_CAPRI_string	26
read_MAF	27
sample_mutations_hist	28
select_genes_on_mutations	28
select_samples_on_mutations	29

<i>annotate_mutational_matrix</i>	3
to_dot	30
update_df	30
Index	32

`annotate_mutational_matrix`
Add samples and genes names to a mutational matrix

Description

Given M mutational matrix, add samples as row names, and genes as column names. If there are repetitions in row names, these are solved by adding a sequential identifier to the names.

Usage

```
annotate_mutational_matrix(M, samples, genes)
```

Arguments

<code>M</code>	mutational matrix
<code>samples</code>	list of sample names
<code>genes</code>	list of gene names

Value

N with the set row and column names

Examples

```
require(Matrix)
genes <- c("A", "B", "C")
samples <- c("S1", "S2", "S2")
M <- Matrix(c(0,0,1,0,0,1,0,1,1), ncol=3, sparse=TRUE, byrow = TRUE)

annotate_mutational_matrix(M, samples, genes)
```

binary_radix_sort *Radix sort for a binary matrix*

Description

Sort the rows of a binary matrix in ascending order

Usage

```
binary_radix_sort(mat)
```

Arguments

mat a binary matrix (of 0 and 1)

Value

the sorted matrix

Examples

```
require(Matrix)
m <- Matrix(c(1,1,0,1,0,0,0,1,1), sparse = TRUE, ncol = 3)
binary_radix_sort(m)
```

build_subset_graph *Remove transitive edges and prepare graph*

Description

Create a graph from the "build_topology_subset" edge list, so that it respects the subset relation, omitting the transitive edges.

Usage

```
build_subset_graph(edges, labels)
```

Arguments

edges edge list, built from "build_topology_subset"
labels list of node labels, to be paired with the graph

Value

a graph with the subset topology, omitting transitive edges

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
edges <- build_topology_subset(samples)
g <- build_subset_graph(edges, labels)
```

build_topology_subset *Compute subset relation as edge list*

Description

Create an edge list E representing the 'subset' relation for binary strings so that:

$$(A, B) \in E \iff \text{forall}(i) : A[i] \geq B[i]$$

Usage

```
build_topology_subset(samples)
```

Arguments

samples input dataset (mutational matrix) as matrix

Value

the computed edge list

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
build_topology_subset(samples)
```

chunk_reader	<i>Gradually read a file from disk</i>
--------------	----------------------------------------

Description

This function creates a reader to read a text file in batches (or chunks). It can be used for very large files that cannot fit in RAM.

Usage

```
chunk_reader(file_path)
```

Arguments

file_path path to large file

Value

a list-object containing the function ‘read’ to read lines from the given file, and ‘close’ to close the connection to the file stream.

Examples

```
# open connection to file
reader <- chunk_reader(
  system.file("extdata", "paac_jhu_2014_500.maf", package = "CIMICE", mustWork = TRUE)
)

while(TRUE){
  # read a chunk
  chunk <- reader$read(10)
  if(length(chunk) == 0){
    break
  }
  # --- process chunk ---
}
# close connection
reader$close()
```

CIMICE

CIMICE Package

Description

R implementation of the CIMICE tool. CIMICE is a tool in the field of tumor phylogenetics and its goal is to build a Markov Chain (called Cancer Progression Markov Chain, CPMC) in order to model tumor subtypes evolution. The input of CIMICE is a Mutational Matrix, so a boolean matrix representing altered genes in a collection of samples. These samples are assumed to be obtained with single-cell DNA analysis techniques and the tool is specifically written to use the peculiarities of this data for the CMPC construction. See <https://github.com/redsnic/tumorEvolutionWithMarkovChains/tree/master/Geno> for the original Java version of this tool.

Details

CIMICE-R: (Markov) Chain Method to Infer Cancer Evolution

Author(s)

Nicolò Rossi <olocin.issor@gmail.com>

compact_dataset

Compact dataset rows

Description

Count duplicate rows and compact the dataset (mutational). The column 'freq' will contain the counts for each row.

Usage

```
compact_dataset(mutmatrix)
```

Arguments

mutmatrix input dataset (mutational matrix)

Value

the compacted dataset (mutational matrix)

Examples

```
compact_dataset(example_dataset())
```

`computeDWNW`*Down weights computation*

Description

Computes the Down weights formula using a Dinamic Programming approach (starting call), see vignettes for further explanation.

Usage

```
computeDWNW(g, freqs, no.of.children, A, normUpWeights)
```

Arguments

<code>g</code>	graph (a Directed Acyclic Graph)
<code>freqs</code>	observed genotype frequencies
<code>no.of.children</code>	number of children for each node
<code>A</code>	adjacency matrix of G
<code>normUpWeights</code>	normalized up weights as computed by <code>normalizeUPW</code>

Value

a vector containing the Up weights for each edge

Examples

```
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
# prepare adj matrix
A <- as.matrix(as_adj(g))
# pre-compute exiting edges from each node
no.of.children <- get_no_of_children(A,g)
upWeights <- computeUPW(g, freqs, no.of.children, A)
normUpWeights <- normalizeUPW(g, freqs, no.of.children, A, upWeights)
computeDWNW(g, freqs, no.of.children, A, normUpWeights)
```

computeDWNW_aux	<i>Down weights computation (aux)</i>
-----------------	---------------------------------------

Description

Computes the Down weights formula using a Dinamic Programming approach (recursion), see vignettes for further explanation.

Usage

```
computeDWNW_aux(g, edge, freqs, no.of.children, A, normUpWeights)
```

Arguments

g	graph (a Directed Acyclic Graph)
edge	the currently considered edge
freqs	observed genotype frequencies
no.of.children	number of children for each node
A	adjacency matrix of G
normUpWeights	normalized up weights as computed by normalizeUPW

Value

a vector containing the Up weights for each edge

computeUPW	<i>Up weights computation</i>
------------	-------------------------------

Description

Computes the up weights formula using a Dinamic Programming approach (starting call), see vignettes for further explanation.

Usage

```
computeUPW(g, freqs, no.of.children, A)
```

Arguments

g	graph (a Directed Acyclic Graph)
freqs	observed genotype frequencies
no.of.children	number of children for each node
A	adjacency matrix of G

Value

a vector containing the Up weights for each edge

Examples

```
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
# prepare adj matrix
A <- as.matrix(as_adj(g))
# pre-compute exiting edges from each node
no.of.children <- get_no_of_children(A,g)
computeUPW(g, freqs, no.of.children, A)
```

computeUPW_aux

Up weights computation (aux)

Description

Computes the up weights formula using a Dinamic Programming approach (recursion), see vignettes for further explanation.

Usage

```
computeUPW_aux(g, edge, freqs, no.of.children, A)
```

Arguments

g	graph (a Directed Acyclic Graph)
edge	the currently considered edge
freqs	observed genotype frequencies
no.of.children	number of children for each node
A	adjacency matrix of G

Value

a vector containing the Up weights for each edge

compute_weights_default
Compute default weights

Description

This procedure computes the weights for edges of a graph accordingly to CIMICE specification. (See vignettes for further explanations)

Usage

```
compute_weights_default(g, freqs)
```

Arguments

g a graph (must be a DAG with no transitive edges)
freqs observed frequencies of genotypes

Value

a graph with the computed weights

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
compute_weights_default(g, freqs)
```

corrplot_from_mutational_matrix
Correlation plot from mutational matrix

Description

Prepare correlation plot based on a mutational matrix

Usage

```
corrplot_from_mutational_matrix(mutmatrix)
```

Arguments

mutmatrix input dataset

Value

the computed correlation plot

Examples

```
corrplot_from_mutational_matrix(example_dataset())
```

corrplot_genes *Gene based correlation plot*

Description

Prepare a correlation plot computed from genes' perspective using a mutational matrix

Usage

```
corrplot_genes(mutmatrix)
```

Arguments

mutmatrix input dataset (mutational matrix)

Value

the computed correlation plot

Examples

```
corrplot_genes(example_dataset())
```

corrplot_samples *Sample based correlation plot*

Description

Prepare a correlation plot computed from samples' perspective using a mutational matrix

Usage

```
corrplot_samples(mutmatrix)
```

Arguments

mutmatrix input dataset (mutational matrix)

Value

the computed correlation plot

Examples

```
corrplot_samples(example_dataset())
```

dataset_preprocessing *Run CIMICE preprocessing*

Description

executes the preprocessing steps of CIMICE

Usage

```
dataset_preprocessing(dataset)
```

Arguments

dataset a mutational matrix as a (sparse) matrix

Details

Preprocessing steps:

- 1) dataset is compacted
- 2) genotype frequencies are computed
- 3) labels are prepared

Value

a list containing the mutational matrix ("samples"), the mutational frequencies of the genotypes ("freqs"), the node labels ("labels") and finally the gene names ("genes")

Examples

```
require(dplyr)
example_dataset() %>% dataset_preprocessing
```

dataset_preprocessing_population

Run CIMICE preprocessing for population format dataset

Description

executes the preprocessing steps of CIMICE

Usage

```
dataset_preprocessing_population(compactDataset)
```

Arguments

compactDataset

a list (matrix: a mutational matrix, counts: number of samples with given genotype). "counts" is normalized automatically.

Details

Preprocessing steps:

- 1) genotype frequencies are computed
- 2) labels are prepared

Value

a list containing the mutational matrix ("samples"), the mutational frequencies of the genotypes ("freqs"), the node labels ("labels") and finally the gene names ("genes")

Examples

```
require(dplyr)
example_dataset_withFreqs() %>% dataset_preprocessing_population
```

draw_ggraph	<i>ggplot graph output</i>
-------------	----------------------------

Description

Draws the output graph using ggplot

Usage

```
draw_ggraph(g, W, labels, digits = 4)
```

Arguments

g	graph to be drawn
W	weights on edges
labels	node labels
digits	precision for edges' weights

Value

ggraph object representing g as described

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
W <- compute_weights_default(g, freqs)
draw_ggraph(g,W,labels,digit = 3)
```

draw_networkD3	<i>NetworkD3 graph output</i>
----------------	-------------------------------

Description

Draws the output graph using networkD3

Usage

```
draw_networkD3(g, W, labels)
```

Arguments

g	graph to be drawn
W	weights on edges
labels	node labels

Value

networkD3 object representing g as described

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
W <- compute_weights_default(g, freqs)
draw_networkD3(g,W,labels)
```

draw_visNetwork	<i>VisNetwork graph output (default)</i>
-----------------	------------------------------------------

Description

Draws the output graph using VisNetwork

Usage

```
draw_visNetwork(g, W, labels)
```

Arguments

g	graph to be drawn
W	weights on edges
labels	node labels

Value

visNetwork object representing g as described

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
W <- compute_weights_default(g, freqs)
draw_visNetwork(g,W,labels)
```

example_dataset	<i>Creates a simple example dataset</i>
-----------------	-----------------------------------------

Description

Creates a simple example dataset

Usage

```
example_dataset()
```

Value

a simple mutational matrix

Examples

```
example_dataset()
```

example_dataset_withFreqs	<i>Creates a simple example dataset with frequency column</i>
---------------------------	---------------------------------------------------------------

Description

Creates a simple example dataset with frequency column

Usage

```
example_dataset_withFreqs()
```

Value

a simple mutational matrix

Examples

```
example_dataset_withFreqs()
```

fix_clonal_genotype *Manage Clonal genotype in data*

Description

Fix the absence of the clonal genotype in the data (if needed)

Usage

```
fix_clonal_genotype(samples, freqs, labels)
```

Arguments

samples	input dataset (mutational matrix) as matrix
freqs	genotype frequencies (in the rows' order)
labels	list of gene names (in the columns' order)

Value

a named list containing the fixed "samples", "freqs" and "labels"

Examples

```
require(dplyr)

# compact
compactDataset <- compact_dataset(example_dataset())
samples <- compactDataset$matrix

# save genes' names
genes <- colnames(compactDataset$matrix)

# keep the information on frequencies for further analysis
freqs <- compactDataset$counts/sum(compactDataset$counts)

# prepare node labels listing the mutated genes for each node
labels <- prepare_labels(samples, genes)

# fix Clonal genotype absence, if needed
fix <- fix_clonal_genotype(samples, freqs, labels)

samples <- fix[["samples"]]
freqs <- fix[["freqs"]]
labels <- fix[["labels"]]
```

```
list("samples" = samples, "freqs" = freqs,  
     "labels" = labels, "genes" = genes)
```

gene_mutations_hist *Histogram of genes' frequencies*

Description

Create the histogram of the genes' mutational frequencies

Usage

```
gene_mutations_hist(mutmatrix, binwidth = 1)
```

Arguments

mutmatrix input dataset (mutational matrix)
binwidth binwidth parameter for the histogram (as in ggplot)

Value

the newly created histogram

Examples

```
gene_mutations_hist(example_dataset(), binwidth = 10)
```

get_no_of_children *Get number of children*

Description

Compute number of children for each node given an adj matrix

Usage

```
get_no_of_children(A, g)
```

Arguments

A Adjacency matrix of the graph g
g a graph

Value

a vector containing the number of children for each node in g

Examples

```
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
A <- as_adj(g)
get_no_of_children(A, g)
```

graph_non_transitive_subset_topology

Default preparation of graph topology

Description

By default, CIMICE computes the relation between genotypes using the subset relation. For the following steps it is also important that the transitive edges are removed.

Usage

```
graph_non_transitive_subset_topology(samples, labels)
```

Arguments

samples	mutational matrix
labels	genotype labels

Value

a graph with the wanted topology

Examples

```
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
graph_non_transitive_subset_topology(samples, labels)
```

make_dataset	<i>Dataset line by line construction: initialization</i>
--------------	----------------------------------------------------------

Description

Initialize a dataset for "line by line" creation

Usage

```
make_dataset(...)
```

Arguments

... gene names (do not use '"', the input is automatically converted to strings)

Value

a mutational matrix without samples structured as (sparse) matrix

Examples

```
make_dataset(APC,P53,KRAS)
```

normalizedDWNW	<i>Down weights normalization</i>
----------------	-----------------------------------

Description

Normalizes Down weights so that the sum of weights of edges exiting a node is 1

Usage

```
normalizedDWNW(g, freqs, no.of.children, A, downWeights)
```

Arguments

g	graph (a Directed Acyclic Graph)
freqs	observed genotype frequencies
no.of.children	number of children for each node
A	adjacency matrix of G
downWeights	Down weights as computed by computeDWNW

Value

a vector containing the normalized Down weights for each edge

Examples

```

require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
# prepare adj matrix
A <- as.matrix(as_adj(g))
# pre-compute exiting edges from each node
no.of.children <- get_no_of_children(A,g)
upWeights <- computeUPW(g, freqs, no.of.children, A)
normUpWeights <- normalizeUPW(g, freqs, no.of.children, A, upWeights)
downWeights <- computeDWNW(g, freqs, no.of.children, A, normUpWeights)
normalizeUPW(g, freqs, no.of.children, A, downWeights)

```

normalizeUPW

Up weights normalization

Description

Normalizes up weights so that the sum of weights of edges entering in a node is 1

Usage

```
normalizeUPW(g, freqs, no.of.children, A, upWeights)
```

Arguments

<code>g</code>	graph (a Directed Acyclic Graph)
<code>freqs</code>	observed genotype frequencies
<code>no.of.children</code>	number of children for each node
<code>A</code>	adjacency matrix of G
<code>upWeights</code>	Up weights as computed by computeUPW

Value

a vector containing the normalized Up weights for each edge

Examples

```
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
# prepare adj matrix
A <- as.matrix(as_adj(g))
# pre-compute exiting edges from each node
no.of.children <- get_no_of_children(A,g)
upWeights <- computeUPW(g, freqs, no.of.children, A)
normalizeUPW(g, freqs, no.of.children, A, upWeights)
```

prepare_labels

Prepare node labels based on genotypes

Description

Prepare node labels so that each node is labelled with a comma separated list of the altered genes representing its associated genotype.

Usage

```
prepare_labels(samples, genes)
```

Arguments

samples	input dataset (mutational matrix) as matrix
genes	list of gene names (in the columns' order)

Details

Note that after this procedure the user is expected also to run `fix_clonal_genotype` to also add the clonal genotype to the mutational matrix if it is not present.

Value

the computed edge list

Examples

```
require(dplyr)

# compact
compactedDataset <- compact_dataset(example_dataset())
samples <- compactedDataset$matrix

# save genes' names
genes <- colnames(compactedDataset$matrix)

# keep the information on frequencies for further analysis
freqs <- compactedDataset$counts/sum(compactedDataset$counts)

# prepare node labels listing the mutated genes for each node
labels <- prepare_labels(samples, genes)
```

quick_run

Run CIMICE defaults

Description

This function executes CIMICE analysis on a dataset using default settings.

Usage

```
quick_run(dataset, mode = "CAPRI")
```

Arguments

dataset	a mutational matrix as a data frame
mode	indicates the used input format. Must be either "CAPRI" or "CAPRIpop"

Value

a list object representing the graph computed by CIMICE with the structure ‘list(topology = g, weights = W, labels = labels)’

Examples

```
quick_run(example_dataset())
```

read	<i>Read a "CAPRI" file</i>
------	----------------------------

Description

Read a "CAPRI" formatted file, as read_CAPRI

Usage

```
read(filepath)
```

Arguments

filepath path to file

Value

the described mutational matrix as a (sparse) matrix

Examples

```
read(system.file("extdata", "example.CAPRI", package = "CIMICE", mustWork = TRUE))
```

read_CAPRI	<i>Read a "CAPRI" file</i>
------------	----------------------------

Description

Read a "CAPRI" formatted file from the file system

Usage

```
read_CAPRI(filepath)
```

Arguments

filepath path to file

Value

the described mutational matrix as a (sparse) matrix

Examples

```
#                    "pathToDataset/myDataset.CAPRI"  
read_CAPRI(system.file("extdata", "example.CAPRI", package = "CIMICE", mustWork = TRUE))
```

read_CAPRIpop	<i>Read a "CAPRIpop" file</i>
---------------	-------------------------------

Description

Read a "CAPRIpop" formatted file from the file system

Usage

```
read_CAPRIpop(filepath)
```

Arguments

filepath	path to file
----------	--------------

Value

a list containing the described mutational matrix as a (sparse) matrix and a list of the frequency of the genotypes

Examples

```
# "pathToDataset/myDataset.CAPRI"  
read_CAPRI(system.file("extdata", "example.CAPRIpop", package = "CIMICE", mustWork = TRUE))
```

read_CAPRI_string	<i>Read "CAPRI" file from a string</i>
-------------------	----------------------------------------

Description

Read a "CAPRI" formatted file, from a text string

Usage

```
read_CAPRI_string(txt)
```

Arguments

txt	string in valid "CAPRI" format
-----	--------------------------------

Value

the described mutational matrix as a (sparse) matrix

Examples

```
read_CAPRI_string("
s\\g A B C D
S1 0 0 0 1
S2 1 0 0 0
S3 1 0 0 0
S4 1 0 0 1
S5 1 1 0 1
S6 1 1 0 1
S7 1 0 1 1
S8 1 1 0 1
")
```

read_MAF

Create mutational matrix from MAF file

Description

Read a MAF (Mutation Annotation Format) file and create a Mutational Matrix combining gene and sample IDs.

Usage

```
read_MAF(path, ...)
```

Arguments

path	path to MAF file
...	other maftools::mutCountMatrix arguments

Value

the mutational (sparse) matrix associated to the MAF file

Examples

```
read_MAF(system.file("extdata", "paac_jhu_2014_500.maf", package = "CIMICE", mustWork = TRUE))
```

sample_mutations_hist *Histogram of samples' frequencies*

Description

Create the histogram of the samples' mutational frequencies

Usage

```
sample_mutations_hist(mutmatrix, binwidth = 1)
```

Arguments

mutmatrix input dataset (mutational matrix)
binwidth binwidth parameter for the histogram (as in ggplot)

Value

the newly created histogram

Examples

```
sample_mutations_hist(example_dataset(), binwidth = 10)
```

select_genes_on_mutations
Filter dataset by genes' mutation count

Description

Dataset filtering on genes, based on their mutation count

Usage

```
select_genes_on_mutations(mutmatrix, n, desc = TRUE)
```

Arguments

mutmatrix input dataset (mutational matrix) to be reduced
n number of genes to be kept
desc TRUE: select the n least mutated genes, FALSE: select the n most mutated genes

Value

the modified dataset (mutational matrix)

Examples

```
# keep information on the 100 most mutated genes
select_genes_on_mutations(example_dataset(), 5)
# keep information on the 100 least mutated genes
select_genes_on_mutations(example_dataset(), 5, desc = FALSE)
```

```
select_samples_on_mutations
      Filter dataset by samples' mutation count
```

Description

Dataset filtering on samples, based on their mutation count

Usage

```
select_samples_on_mutations(mutmatrix, n, desc = TRUE)
```

Arguments

mutmatrix	input dataset (mutational matrix) to be reduced
n	number of samples to be kept
desc	T: select the n least mutated samples, F: select the n most mutated samples

Value

the modified dataset (mutational matrix)

Examples

```
require(dplyr)
# keep information on the 5 most mutated samples
select_samples_on_mutations(example_dataset(), 5)
# keep information on the 5 least mutated samples
select_samples_on_mutations(example_dataset(), 5, desc = FALSE)
# combine selections
select_samples_on_mutations(example_dataset() , 5, desc = FALSE) %>%
  select_genes_on_mutations(5)
```

to_dot	<i>Dot graph output</i>
--------	-------------------------

Description

Represents this graph in dot format (a textual output format)

Usage

```
to_dot(g, W, labels)
```

Arguments

<code>g</code>	graph to be drawn
<code>W</code>	weights on edges
<code>labels</code>	node labels

Value

a string representing the graph in dot format

Examples

```
require(dplyr)
require(purrr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
genes <- preproc[["genes"]]
g <- graph_non_transitive_subset_topology(samples, labels)
W <- compute_weights_default(g, freqs)
to_dot(g,W,labels)
```

update_df	<i>Dataset line by line construction: add a sample</i>
-----------	--------------------------------------------------------

Description

Add a sample (a row) to an existing dataset. This procedure is meant to be used with the "

Usage

```
update_df(mutmatrix, sampleName, ...)
```

Arguments

<code>mutmatrix</code>	an existing (sparse) matrix (mutational matrix)
<code>sampleName</code>	the row (sample) name
<code>...</code>	sample's genotype (0/1 numbers)

Value

the modified (sparse) matrix (mutational matrix)

Examples

```
require(dplyr)
make_dataset(APC,P53,KRAS) %>%
  update_df("S1", 1, 0, 1) %>%
  update_df("S2", 1, 1, 1)
```

Index

[annotate_mutational_matrix](#), [3](#)

[binary_radix_sort](#), [4](#)
[build_subset_graph](#), [4](#)
[build_topology_subset](#), [5](#)

[chunk_reader](#), [6](#)
[CIMICE](#), [7](#)
[compact_dataset](#), [7](#)
[compute_weights_default](#), [11](#)
[computeDWNW](#), [8](#)
[computeDWNW_aux](#), [9](#)
[computeUPW](#), [9](#)
[computeUPW_aux](#), [10](#)
[corrplot_from_mutational_matrix](#), [11](#)
[corrplot_genes](#), [12](#)
[corrplot_samples](#), [13](#)

[dataset_preprocessing](#), [13](#)
[dataset_preprocessing_population](#), [14](#)
[draw_ggraph](#), [15](#)
[draw_networkD3](#), [15](#)
[draw_visNetwork](#), [16](#)

[example_dataset](#), [17](#)
[example_dataset_withFreqs](#), [17](#)

[fix_clonal_genotype](#), [18](#)

[gene_mutations_hist](#), [19](#)
[get_no_of_children](#), [19](#)
[graph_non_transitive_subset_topology](#),
[20](#)

[make_dataset](#), [21](#)

[normalizeDWNW](#), [21](#)
[normalizeUPW](#), [22](#)

[prepare_labels](#), [23](#)

[quick_run](#), [24](#)

[read](#), [25](#)
[read_CAPRI](#), [25](#)
[read_CAPRI_string](#), [26](#)
[read_CAPRIpop](#), [26](#)
[read_MAF](#), [27](#)

[sample_mutations_hist](#), [28](#)
[select_genes_on_mutations](#), [28](#)
[select_samples_on_mutations](#), [29](#)

[to_dot](#), [30](#)

[update_df](#), [30](#)