Package 'CIMICE'

May 8, 2024

Type Package

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

Version 1.12.0

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, ggcorrplot, purrr, ggraph, stats, utils, maftools, assertthat, tidygraph, expm, Matrix

RoxygenNote 7.1.2

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_19

git_last_commit 5c11705

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-08

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>

Contents

`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

`M`	mutational matrix
`samples`	list of sample names
`genes`	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:

$$(A, B) inE <=> forall(i) : A[i] -> B[i]$$

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 *Gradually read a file from disk*

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/Genc 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

a list with matrix (the compacted dataset (mutational matrix)), counts (frequencies of genotypes) and row_names (comma separated string of sample IDs) fields

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 explaination.

Usage

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

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
normUpWeights	normalized up weights as computed by normalizeUPW

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 *Down weights computation (aux)*

Description

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

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 *Up weights computation*

Description

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

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 explaination.

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 explainations)

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 poulation format dataset*

Description

executes the preprocessing steps of CIMICE

Usage

```
dataset_preprocessing_population(compactedDataset)
```

Arguments

`compactedDataset`
: 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 *ggplot graph output*

Description

Draws the output graph using ggplot

Usage

```
draw_ggraph(out, digits = 4, ...)
```

Arguments

out	the output object of CIMICE (es, from quick run)
digits	precision for edges' weights
...	other arguments for format_labels

Value

ggraph object representing g as described

Examples

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

draw_networkD3 *NetworkD3 graph output*

Description

Draws the output graph using networkD3

Usage

```
draw_networkD3(out, ...)
```

Arguments

out	the output object of CIMICE (es, from quick run)
...	other arguments for format_labels

Value

networkD3 object representing g as described

Examples

```
draw_networkD3(quick_run(example_dataset()))
```

---

draw_visNetwork *VisNetwork graph output (default)*

Description

Draws the output graph using VisNetwork

Usage

```
draw_visNetwork(out, ...)
```

Arguments

out	the output object of CIMICE (es, from quick run)
...	other arguments for format_labels

Value

visNetwork object representing g as described

Examples

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

---

example_dataset *Creates a simple example dataset*

Description

Creates a simple example dataset

Usage

```
example_dataset()
```

Value

a simple mutational matrix

Examples

```
example_dataset()
```

example_dataset_withFreqs *Creates a simple example dataset with frequency column*

Description

Creates a simple example dataset with frequency column

Usage

```
example_dataset_withFreqs()
```

Value

a simple mutational matrix

Examples

```
example_dataset_withFreqs()
```

finalize_generator *Finalize generator normalizing edge weights*

Description

Checks if a generator can be normalized so that it actually is a Markov Chain

Usage

```
finalize_generator(generator)
```

Arguments

generator	a generator

Value

A generator with edge weights that respect DTMC definition

Examples

```
require(dplyr)

example_dataset() %>%
 make_generator_stub() %>%
 set_generator_edges(
   list(
    "D", "A, D", 1 ,
    "A", "A, D", 1 ,
    "A, D", "A, C, D", 1 ,
    "A, D", "A, B, D", 1 ,
    "Clonal", "D", 1 ,
    "Clonal", "A", 1 ,
    "D", "D", 1 ,
    "A", "A", 1 ,
    "A, D", "A, D", 1 ,
    "A, C, D", "A, C, D", 1 ,
    "A, B, D", "A, B, D", 1 ,
    "Clonal", "Clonal", 1
  )) %>%
  finalize_generator
```

---

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, matching_samples)
```

Arguments

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

Value

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

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)
if( is.null(compactedDataset$row_names) ){
  compactedDataset$row_names <- rownames(compactedDataset$matrix)
}
matching_samples <- compactedDataset$row_names
# matching_samples
matching_samples

# fix Colonal genotype absence, if needed
fix <- fix_clonal_genotype(samples, freqs, labels, matching_samples)
```

---

format_labels *Format labels for output object*

Description

Prepare labels based on multiple identifiers so that they do not excede a certain size (if they do, a simple number is used)

Usage

```
format_labels(labels, max_col = 3, max_row = 3)
```

Arguments

labels	a charachter vector of the labels to manage
max_col	maximum number of identifiers in a single row for a label
max_row	maximum number of rows of identifiers in a label

Value

the updated labels

Examples

```
format_labels(c("A, B", "C, D, E"))
```

`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 *Dataset line by line construction: initialization*

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)
```

make_generator_stub *Create a stub for a generator*

Description

Create a generator topology directly from a dataset. The topology will follow the subset relation.

Usage

```
make_generator_stub(dataset)
```

Arguments

dataset	A compacted CIMICE dataset

Value

a generator, with weight = 0 for all the edges

Examples

```
make_generator_stub(example_dataset())
```

make_labels *Helper function to create labels*

Description

This function helps creating labels for nodes with different information

Usage

```
make_labels(out, mode = "samplesIDs", max_col = 3, max_row = 3)
```

Arguments

out	the output object of CIMICE (es, from quick run)
mode	which labels to print: samplesIDs (matching samples), sequentialIDs (just a sequential numer), geneIDs (genotype)
max_col	identifiers are represented in a max_col times max_row grid (if the number of IDs exceeds, a the sequentialID number is used instead)
max_row	identifiers are represented in a max_col times max_row grid (if the number of IDs exceeds, a the sequentialID number is used instead)

Value

the requested labels

Examples

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

normalizeDWNW *Down weights normalization*

Description

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

Usage

```
normalizeDWNW(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

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

---

perturb_dataset *Perturbate a boolean matrix*

Description

Given a boolean matrix, randomly add False Positives (FP), False Negatives (FN) and Missing data following user defined rates. In the final matrix, missing data is represented by the value 3.

Usage

```
perturb_dataset(dataset, FP_rate = 0, FN_rate = 0, MIS_rate = 0)
```

Arguments

dataset	a matrix/sparse matrix
FP_rate	False Positive rate
FN_rate	False Negative rate
MIS_rate	Missing Data rate

Details

Note that CIMICE does not support dataset with missing data natively, so using MIS_rate != 0 will then require some pre-processing.

Value

the new, perturbed, matrix

Examples

```
require(dplyr)

example_dataset() %>%
  make_generator_stub() %>%
  set_generator_edges(
    list(
      "D", "A, D", 1 ,
      "A", "A, D", 1 ,
      "A, D", "A, C, D", 1 ,
      "A, D", "A, B, D", 1 ,
      "Clonal", "D", 1 ,
      "Clonal", "A", 1 ,
      "D", "D", 1 ,
      "A", "A", 1 ,
      "A, D", "A, D", 1 ,
      "A, C, D", "A, C, D", 1 ,
      "A, B, D", "A, B, D", 1 ,
      "Clonal", "Clonal", 1
  )) %>%
  finalize_generator %>%
  simulate_generator(3, 10) %>%
  perturb_dataset(FP_rate = 0.01, FN_rate = 0.1, MIS_rate = 0.12)
```

---

plot_generator *Plot a generator*

Description

Simple ggraph interface to draw a generator

Usage

```
plot_generator(generator)
```

Arguments

generator	a generator

Value

a basic plot of this generator

Examples

```
require(dplyr)

example_dataset() %>%
 make_generator_stub() %>%
 set_generator_edges(
   list(
    "D", "A, D", 1 ,
    "A", "A, D", 1 ,
    "A, D", "A, C, D", 1 ,
    "A, D", "A, B, D", 1 ,
    "Clonal", "D", 1 ,
    "Clonal", "A", 1 ,
    "D", "D", 1 ,
    "A", "A", 1 ,
    "A, D", "A, D", 1 ,
    "A, C, D", "A, C, D", 1 ,
    "A, B, D", "A, B, D", 1 ,
    "Clonal", "Clonal", 1
  )) %>%
  finalize_generator %>%
  plot_generator
```

---

prepare_generator_edge_set_command

Prepare a command to add edge weights to a generator

Description

Prints a string in the form of the command that sets weights for all the edges of this generator.

Usage

```
prepare_generator_edge_set_command(generator, by = "labels")
```

Arguments

generator	a generator
by	"labels" or "samples" to use gene labels or sample labels as references for edge identifiers.

Value

NULL (the string with the function calls is printed on the stdout)

Examples

```
require(dplyr)
example_dataset() %>%
  make_generator_stub() %>%
  prepare_generator_edge_set_command()
```

`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 alterated 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 genortype 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 (sparse) matrix
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, freqs=freqs)‘

Examples

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

read *Read a "CAPRI" file*

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 *Read a "CAPRI" file*

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 *Read a "CAPRIpop" file*

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_CAPRIpop_string` *Read "CAPRIpop" file from a string*

Description

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

Usage

```
read_CAPRIpop_string(txt)
```

Arguments

`txt`	string in valid "CAPRIpop" format

Value

the described mutational matrix as a (sparse) matrix

Examples

```
read_CAPRIpop_string("
s\\g A B C D freqs
S1 0 0 0 1 0.1
S2 1 0 0 0 0.1
S3 1 0 0 0 0.2
S4 1 0 0 1 0.3
S5 1 1 0 1 0.05
S6 1 1 0 1 0.1
S7 1 0 1 1 0.05
S8 1 1 0 1 0.01
")
```

read_CAPRI_string *Read "CAPRI" file from a string*

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))
```

---

read_matrix *Read dataset from an R matrix*

Description

also converts that matrix to a sparse matrix

Usage

```
read_matrix(mat)
```

Arguments

mat	a boolean mutational matrix

Value

the sparse mutational matrix to be used as CIMICE's input

Examples

```
m <- matrix(c(0,0,1,1,0,1,1,1,1), ncol=3)
colnames(m) <- c("A","B","C")
rownames(m) <- c("S1", "S2", "S3")
read_matrix(m)
```

remove_transitive_edges *Remove transitive edges from an edgelist*

Description

Remove transitive edges from an edgelist. This procedure is temporary to cover a bug in 'relations' package.

Usage

```
remove_transitive_edges(E)
```

Arguments

E	edge list, built from "build_topology_subset"

Value

a new edgelist without transitive edges (as a N*2 matrix)

Examples

```
l <- list(c(1,2),c(2,3), c(1,3))
remove_transitive_edges(l)
```

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)
```

---

set_generator_edges *Add edge weights to a generator*

Description

Add edge weights to a generator

Usage

```
set_generator_edges(generator, f_t_w_list, by = "labels")
```

Arguments

generator	a generator
f_t_w_list	a list of triplets (from, to, list), the triplets must not be nested in the list. For example list("A","B",0.3, "B", "C", 0.2) is a valid input.
by	"labels" or "samples" to use gene labels or sample labels as references for edge identifiers.

Value

the generator with the modified edges (invalid edges are ignored)

Examples

```
require(dplyr)

example_dataset() %>%
 make_generator_stub() %>%
 set_generator_edges(
   list(
    "D", "A, D", 1 ,
    "A", "A, D", 1 ,
    "A, D", "A, C, D", 1 ,
    "A, D", "A, B, D", 1 ,
    "Clonal", "D", 1 ,
    "Clonal", "A", 1 ,
    "D", "D", 1 ,
    "A", "A", 1 ,
    "A, D", "A, D", 1 ,
    "A, C, D", "A, C, D", 1 ,
    "A, B, D", "A, B, D", 1 ,
    "Clonal", "Clonal", 1
  ))
```

---

simulate_generator *Create datasets from generators*

Description

Simulate the DTMC associated to the generator to create a dataset that reflects the genotypes of ‘times‘ cells, sampled after ‘time_ticks‘ passages.

Usage

```
simulate_generator(
  generator,
  time_ticks,
  times,
  starting_label = "Clonal",
  by = "labels",
  mode = "full"
)
```

Arguments

generator	a generator
time_ticks	number of steps (updates) of the DTMC associated to the generato
times	number of sumlated cells
starting_label	node from which to start the simulation

by	"labels" or "samples" to use gene labels or sample labels as references to identify the ‘starting_label‘’s node
mode	"full" to generate a matrix with ‘times‘ genotypes, "compacted" to *efficiently* create an already compacted dataset (a dataset showing the genotypes and their respective frequencies)

Value

the simulated dataset

Examples

```
require(dplyr)

example_dataset() %>%
  make_generator_stub() %>%
  set_generator_edges(
    list(
      "D", "A, D", 1 ,
      "A", "A, D", 1 ,
      "A, D", "A, C, D", 1 ,
      "A, D", "A, B, D", 1 ,
      "Clonal", "D", 1 ,
      "Clonal", "A", 1 ,
      "D", "D", 1 ,
      "A", "A", 1 ,
      "A, D", "A, D", 1 ,
      "A, C, D", "A, C, D", 1 ,
      "A, B, D", "A, B, D", 1 ,
      "Clonal", "Clonal", 1
  )) %>%
  finalize_generator %>%
  simulate_generator(3, 10)
```

---

to_dot *Dot graph output*

Description

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

Usage

```
to_dot(out, ...)
```

Arguments

out	the output object of CIMICE (es, from quick run)
...	other arguments for format_labels

Value

a string representing the graph in dot format

Examples

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

`update_df` *Dataset line by line construction: add a sample*

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

`mutmatrix`	an existing (sparse) matrix (mutational matrix)
`sampleName`	the row (sample) name
`...`	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