Package ‘CIMICE’

December 5, 2023

Type Package
Title CIMICE-R: (Markov) Chain Method to Infer Cancer Evolution
Version 1.10.0
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_18
git_last_commit ff02f0e
git_last_commit_date 2023-10-24
Repository Bioconductor 3.18
Date/Publication  2023-12-05
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 ........................................................... 16
example_dataset_withFreqs ........................................... 17
finalize_generator ......................................................... 17
fix_clonal_genotype ......................................................... 18
format_labels ............................................................... 19
gene_mutations_hist ....................................................... 20
get_no_of_children ......................................................... 20
graph_non_transitive_subset_topology ................................ 21
make_dataset ............................................................... 22
make_generator_stub ......................................................... 22
make_labels ............................................................... 23
normalizeDWNW ............................................................ 23
normalizeUPW ............................................................. 24
perturb_dataset ............................................................ 25
plot_generator ............................................................. 26
prepare_generator_edge_set_command .................................... 27
prepare_labels ............................................................ 28
quick_run ............................................................... 29
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)
genres <- 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, genres)
```

---

**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

```r
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)
```

Description

Create an edge list \( E \) representing the ‘subset’ relation for binary strings so that:

\[ (A, B) \in E \iff \forall (i): A[i] \geq B[i] \]

Usage

`build_topology_subset(samples)`

Arguments

- **samples**: input dataset (mutational matrix) as matrix

Value

the computed edge list

Examples

```r
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 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 CPMC construction. See ‘https://github.com/redsnic/tumorEvolutionWithMarkovChains/tree/master/GenotypeEvolutionPaths’ 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 Dynamic Programming approach (starting call), see vignettes for further explanation.

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

```r
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[\"samples\"]
freqs <- preproc[\"freqs\"]
labels <- preproc[\"labels\"]
genenes <- 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 Dynamic 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
Up weights computation

Description
Computes the up weights formula using a Dynamic 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
computeUPW_aux

Value

a vector containing the Up weights for each edge

Examples

```r
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)
```

Description

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

Usage

```r
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**

```r
require(dplyr)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[['samples']]
freqs <- preproc[['freqs']]labels <- preproc[['labels']]
genesis <- 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(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

```r
require(dplyr)
exmpale_dataset() %>% dataset_preprocessing
```

Description

executes the preprocessing steps of CIMICE

Usage

```r
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

```r
require(dplyr)
exmpale_dataset_withFreqs() %>% dataset_preprocessing_population
```
**draw_ggraph**

*ggplot graph output*

**Description**

Draws the output graph using `ggplot`

**Usage**

```r
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**

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

---

**draw_networkD3**

*NetworkD3 graph output*

**Description**

Draws the output graph using `networkD3`

**Usage**

```r
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
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

```r
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
```

Value

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

Description

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

Usage

```r
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

```r
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$matrix)
}
mapping_samples <- compactedDataset$row_names

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

---

**format_labels**

Format labels for output object

**Description**

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

**Usage**

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

**Arguments**

- `labels` a character 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
get_no_of_children

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

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

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

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

Arguments

- `samples`: mutational matrix
- `labels`: genotype labels

Value

A graph with the wanted topology

Examples

```r
require(dplyr)
preadoc <- example_dataset() %>% dataset_preprocessing
samples <- predoc[["samples"]
freqs <- predoc[["freqs"]
labels <- predoc[["labels"]
gen <- predoc[["genes"]
g <- graph_non_transitive_subset_topology(samples, labels)
```
**Dataset line by line construction: initialization**

**Description**

Initialize a dataset for "line by line" creation

**Usage**

```r
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**

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

---

**Create a stub for a generator**

**Description**

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

**Usage**

```r
make_generator_stub(dataset)
```

**Arguments**

- `dataset` A compacted CIMICE dataset

**Value**

a generator, with weight = 0 for all the edges

**Examples**

```r
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, 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, 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)
```
normalizeUPW

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
perturb_dataset

Value

a vector containing the normalized Up weights for each edge

Examples

```r
require(dplyr)
require(igraph)
preproc <- example_dataset() %>% dataset_preprocessing
samples <- preproc[["samples"]]
freqs <- preproc[["freqs"]]
labels <- preproc[["labels"]]
gen - 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

```r
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
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

```r
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
```

Description

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

Usage

```r
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

```r
require(dplyr)
ex_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 altered genes representing its associated genotype.

Usage

```r
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

```r
require(dplyr)

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

genes <- colnames(compactedDataset$matrix)

defs <- compactedDataset$counts/sum(compactedDataset$counts)

defs <- 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

```r
# "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**

```r
read_CAPRIpop_string(txt)
```

**Arguments**

- `txt`  
  string in valid "CAPRIpop" format

**Value**

the described mutational matrix as a (sparse) matrix

**Examples**

```r
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, ...)

---
**read_matrix**

**Arguments**

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

**Value**

the mutational (sparse) matrix associated to the MAF file

**Examples**

```r
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**

```r
read_matrix(mat)
```

**Arguments**

- `mat`  
  a boolean mutational matrix

**Value**

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

**Examples**

```r
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_topoogy_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
**select_genes_on_mutations**

*Filter dataset by genes’ mutation count*

**Description**

Dataset filtering on genes, based on their mutation count

**Usage**

```r
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**

```r
# 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**

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

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

```r
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

```r
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 generator
- `times`: number of simulated 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

```r
require(dplyr)

elementary_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)
```

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
**update_df**

**Value**

a string representing the graph in dot format

**Examples**

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

---

**Description**

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

**Usage**

```r
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**

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

annotate_mutation_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_mutation_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, 16
example_dataset_withFreqs, 17
finalize_generator, 17
fix_clonal_genotype, 18
format_labels, 19
gene_mutations_hist, 20
gene_no_of_children, 20
graph_non_transitive_subset_topology, 21
make_dataset, 22
make_generator_stub, 22
make_labels, 23
normalizeDWNW, 23
normalizeUPW, 24
perturb_dataset, 25
plot_generator, 26
prepare_generator_edge_set_command, 27
prepare_labels, 28
quick_run, 29
read, 29
read_CAPRI, 30
read_CAPRI_string, 32
read_CAPRIpop, 30
read_CAPRIpop_string, 31
read_MAF, 32
read_matrix, 33
remove_transitive_edges, 34
sample_mutations_hist, 34
select_genes_on_mutations, 35
select_samples_on_mutations, 35
set_generator_edges, 36
simulate_generator, 37
to_dot, 38
update_df, 39