Package ‘gRbase’

April 2, 2022

Version 1.8.7
Title A Package for Graphical Modelling in R
Author Søren Højsgaard <sorenh@math.aau.dk>
Maintainer Søren Højsgaard <sorenh@math.aau.dk>
Description The 'gRbase' package provides graphical modelling features
used by e.g. the packages 'gRain', 'gRim' and 'gRc'. 'gRbase' implements
graph algorithms including (i) maximum cardinality search (for marked
and unmarked graphs).
(ii) moralization, (iii) triangulation, (iv) creation of junction tree.
'gRbase' facilitates array operations,
'gRbase' implements functions for testing for conditional independence.
'gRbase' illustrates how hierarchical log-linear models may be
implemented and describes concept of graphical meta
data.
The facilities of the package are documented in the book by Højsgaard,
Edwards and Lauritzen (2012,
<doi:10.1007/978-1-4614-2299-0>) and in the paper by
Please see 'citation("gRbase")' for citation details.
NOTICE 'gRbase' requires that the packages graph,
'Rgraphviz' and 'RBGL' are installed from 'bioconductor'; for
installation instructions please refer to the web page given below.
License GPL (>= 2)
URL https://people.math.aau.dk/~sorenh/software/gR/
ByteCompile Yes
Encoding UTF-8
VignetteBuilder knitr
Depends R (>= 3.6.0), methods
Imports BiocManager, graph, RBGL, Rgraphviz, stats4, igraph, magrittr,
Matrix, Rcpp (>= 0.11.1)
Suggests testthat (>= 2.1.0), microbenchmark, knitr
LinkingTo Rcpp (>= 0.11.1), RcppEigen, RcppArmadillo
**RoxygenNote 7.1.2**
**NeedsCompilation** yes
**Repository** CRAN
**Date/Publication** 2022-04-02 12:30:02 UTC

**R topics documented:**

all_pairs .................................................. 3
all_subsets ............................................... 4
api-array-07 .............................................. 4
api-array-properties ...................................... 6
api-cell ..................................................... 7
api-cell_ ................................................... 8
api-parray .................................................. 9
api-pct-operations ...................................... 11
api-tabDist ............................................... 12
api-tabNew ................................................ 14
api-tabX .................................................... 15
api-tabX_ .................................................. 16
api_tabSlice ............................................... 17
array-simulate ............................................ 18
compareModels ............................................ 20
cov2pcor ................................................... 20
dag2chol ..................................................... 21
data-ashtrees .............................................. 22
data-BodyFat ................................................ 23
data-breastcancer ........................................ 24
data-cad ..................................................... 25
data-carcass .............................................. 26
data-chestSim .............................................. 27
data-dietox ............................................... 28
data-dumping ............................................... 29
data-lizard .................................................. 29
data-mathmark .......................................... 30
data-mildew ............................................... 31
data-milkcomp ............................................ 32
data-Nutrimouse .......................................... 33
data-personality ......................................... 34
data-rats .................................................... 35
data-reinis .................................................. 35
data-wine .................................................... 36
downstream-aliases ...................................... 37
fastcombn ................................................. 37
gmwr_book .................................................. 38
graph-clique ............................................... 39
graph-coerce .............................................. 40
graph-coerce-api ......................................... 41
**Description**

Create all possible pairs of two character vectors.

**Usage**

```r
all_pairs(x, y = character(0), sort = FALSE, result = "matrix")
```

```r
names2pairs(x, y = NULL, sort = TRUE, result = "list")
```

**Arguments**

- `x, y` Character vectors.
- `sort` Logical.
- `result` A list or a matrix.

**Details**

NOTICE: If `y` is not `NULL` then `x` and `y` must be disjoint (no checks are made); otherwise pairs of identical elements will also be obtained.
Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
x <- letters[1:4]
y <- letters[5:7]

all_pairs(x)
all_pairs(x, result="matrix")

all_pairs(x, y)
all_pairs(x, y, result="matrix")
```

---

all_subsets Create all subsets

Description

Create all subsets of a vector

Usage

```r
all_subsets(x)
all_subsets0(x)
```

Arguments

- `x` Vector

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

---

api-array-07 Array operations (2007)

Description

Array operations; created to facilitate the gRain package in 2007. Now largely replaceable by other (often faster) functions implemented in Rcpp.
Usage

```
  tablePerm(tab, perm, resize = TRUE, keep.class = FALSE)
  tableMult(tab1, tab2)
  tableDiv(tab1, tab2)
  tableOp(tab1, tab2, op = "*")
  tableOp2(tab1, tab2, op = "/", restore = FALSE)
  tableOp0(tab1, tab2, op = "/")
  tableSlice(tab, margin, level, impose)
  tableSlicePrim(tab, mar.idx, lev.idx)
  tableMargin(tab, margin, keep.class = FALSE)
  tableGetSliceIndex(tab, margin, level, complement = FALSE)
  tableSetSliceValue(tab, margin, level, complement = FALSE, value = 0)
```

Arguments

- `tab`, `tab1`, `tab2`: Arrays with named dimnames.
- `perm`: A permutation; either indices or names.
- `resize`: A flag indicating whether the vector should be resized as well as having its elements reordered (default TRUE).
- `keep.class`: Obsolete argument.
- `op`: The operation; choices are "*", "/", "#", "-".
- `restore`: Not so clear anymore.
- `margin`: Index or name of margin.
- `level`: Corresponding level of margin.
- `impose`: Value to be imposed.
- `mar.idx`: Index of margin
- `lev.idx`: Index of level
- `complement`: Should values be set for the complement?
- `value`: Which value should be set

Details

`tableOp0` is brute force implementation based on dataframes. It is very slow, but useful for error checking.
Check if object is array

Description

Check if object is array (that it is a vector with a dim attribute) and that the object has dimnames and that dimnames are named.

Usage

```r
is.named.array(obj)

is_named_array_(obj)

is_number_vector_(obj)

is_dimnames_(obj)

dimnames_match(a1, a2)
```

Arguments

- `obj` Some R object.
- `a1, a2` Arrays with named dimnames.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
is.named.array( HairEyeColor )
is.named.array( matrix(1:4, nrow=2) )
is_named_array_( HairEyeColor )
is_named_array_( matrix(1:4, nrow=2) )
is_number_vector_(1:4)
is_number_vector_((list(1:4))

ar1 = tabNew(c("a", "b"), levels=c(2, 3))
ar2 = tabNew(c("c", "a"), levels=c(2, 2))
ar1

ar2
## dimension a has levels a1,a2 in both ar1 and ar2. # Hence we have a match.
dimnames_match(ar1, ar2)

ar1 = tabNew(c("a", "b"), levels=c(2, 3))
ar2 = tabNew(c("c", "a"), levels=c(2, 3))
ar1
```
## dimension a has levels a1,a2 in ar1 and levels a1,a2,a3 in ar2.
# Hence we do not have a match.
dimnames_match(ar1, ar2)

ar2 = tabNew(c("c", "a"), levels=list(c=c("c1", "c2"), a=c("a2", "a1")))

## dimension a has levels a1,a2 in ar1 and levels a2,a1 in ar2.
# Hence we do not have a match.
dimnames_match(ar1, ar2)

---

**Table cell operations.**

**Description**

Low level table cell operations.

**Usage**

- `cell2entry(cell, dim)`
- `entry2cell(entry, dim)`
- `next_cell(cell, dim)`
- `next_cell2(cell, dim)`
- `next_cell_slice(cell, dim, slice_marg)`
- `slice2entry(slice_cell, slice_marg, dim)`
- `cell2entry_perm(cell, dim, perm)`
- `perm_cell_entries(perm, dim)`
- `fact_grid(dim, slice_cell = NULL, slice_marg = NULL)`

**Arguments**

- `cell` Vector giving the cell, e.g. c(1, 1, 2) in 3-way table.
- `dim` Vector giving array dimension, eg c(2, 2, 2).
- `entry` An entry in an array (a number indexing a vector).
- `slice_marg` Vector giving the margin of a table, eg. c(2, 3)
- `slice_cell` Vector giving the corresponding cell of marginal table, e.g. c(1, 2)
- `perm` Vector giving permutaion of array, eg. c(1, 3, 2).
Examples

di <- c(2, 2, 3)
cell2entry(c(1, 1, 1), dim=di)
cell2entry(c(2, 2, 3), dim=di)

entry2cell(1, dim=di)
entry2cell(12, dim=di)

next_cell(c(1, 1, 1), dim=di)
next_cell(c(2, 1, 1), dim=di)

## The first two entries are kept fixed
next_cell_slice(c(2, 1, 1), dim=di, slice_marg=c(1, 2))
next_cell_slice(c(2, 1, 2), dim=di, slice_marg=c(1, 2))

## Cell (2, 2, 1) corresponds to entry 4
cell2entry(c(2, 2, 1), dim=di)
## Same as
cell2entry_perm(c(2, 2, 1), dim=di, perm=c(1, 2, 3))
## If the table dimensions are permuted as (3, 1, 2)
## the entry becomes
cell2entry_perm(c(2, 2, 1), dim=di, perm=c(3, 1, 2))

---

Description

Corresponding R functions without the trailing underscore exist.

Usage

cell2entry_(cell, dim)
make_plevels_(dim)
entry2cell_(entry, dim)
next_cell_(cell, dim)
next_cell2_(cell, dim)
next_cell_slice_(cell, dim, slice_marg)
slice2entry_(slice_cell, slice_marg, dim)
cell2entry_perm_ (cell, dim, perm)

perm_cell_entries_ (perm, dim)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>Vector giving the cell, e.g. c(1, 1, 2) in 3-way table.</td>
</tr>
<tr>
<td>dim</td>
<td>Vector giving array dimension, eg c(2, 2, 2).</td>
</tr>
<tr>
<td>entry</td>
<td>An entry in an array (a number indexing a vector).</td>
</tr>
<tr>
<td>slice_marg</td>
<td>Vector giving the margin of a table, eg. c(2, 3)</td>
</tr>
<tr>
<td>slice_cell</td>
<td>Vector giving the corresponding cell of marginal table, e.g. c(1, 2)</td>
</tr>
<tr>
<td>perm</td>
<td>Vector giving permutation of array, eg. c(1, 3, 2).</td>
</tr>
</tbody>
</table>

Description

General representation of multidimensional arrays (with named dimnames, also called named arrays.)

Usage

parray(varNames, levels, values = 1, normalize = "none", smooth = 0)
as.parray(values, normalize = "none", smooth = 0)
data2parray(data, varNames = NULL, normalize = "none", smooth = 0)
makeDimNames(varNames, levels, sep = "")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varNames</td>
<td>Names of variables defining table; can be a right hand sided formula.</td>
</tr>
<tr>
<td>levels</td>
<td>Either 1) a vector with number of levels of the factors in varNames or 2) a</td>
</tr>
<tr>
<td></td>
<td>list with specification of the levels of the factors in varNames. See 'examples' below.</td>
</tr>
<tr>
<td>values</td>
<td>Values to go into the array</td>
</tr>
<tr>
<td>normalize</td>
<td>Either &quot;none&quot;, &quot;first&quot; or &quot;all&quot;. Should result be normalized, see 'Details' below.</td>
</tr>
<tr>
<td>smooth</td>
<td>Should values be smoothed, see 'Details' below.</td>
</tr>
<tr>
<td>data</td>
<td>Data to be coerced to a parray; can be data.frame, table, xtabs, matrix.</td>
</tr>
<tr>
<td>sep</td>
<td>Desired separator in dim names; defaults to &quot;&quot;.</td>
</tr>
</tbody>
</table>
Details

A named array object represents a table defined by a set of variables and their levels, together with
the values of the table. E.g. f(a,b,c) can be a table with a,b,c representing levels of binary variable

If `normalize="first"` then for each configuration of all other variables than the first, the proba-
bilities are normalized to sum to one. Thus f(a,b,c) becomes a conditional probability table of the
form p(abc).

If `normalize="all"` then the sum over all entries of f(a,b,c) is one.

If `smooth` is positive then `smooth` is added to `values` before normalization takes place.

Value

A a named array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`is.named.array`

Examples

t1 <- parray(c("gender","answer"), list(c('male','female'),c('yes','no')), values=1:4)
t1 <- parray(~gender:answer, list(c('male','female'),c('yes','no')), values=1:4)
t1 <- parray(~gender:answer, c(2,2), values=1:4)

t2 <- parray(c("answer","category"), list(c('yes','no'),c(1,2)), values=1:4+10)
t3 <- parray(c("category","foo"), c(2,2), values=1:4+100)

varNames(t1)
nLevels(t1)
valueLabels(t1)

## Create 1-dimensional vector with dim and dimnames
x1 <- 1:5
as.parray(x1)
x2 <- parray("x", levels=length(x1), values=x1)
dim(x2)
dimnames(x2)

## Matrix
x1 <- matrix(1:6, nrow=2)
as.parray(x1)
parray(~a:b, levels=dim(x1), values=x1)

## Extract pararrays from data
## 1) a dataframe
data(cad1)
data2parray(cad1, "Sex:AngPec:AMI")
data2parray(cad1, c("Sex","AngPec","AMI"))
data2parray(cad1, c(1,2,3))
## 2) a table
data2parray(UCBAAdmissions,c(1,2), normalize="first")

---

**api-pct-operations**  
**Array algebra**

**Description**  
Addition, subtraction etc. of arrays

**Usage**

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a1 %a+% a2</code></td>
<td>Addition of arrays <code>a1</code> and <code>a2</code></td>
</tr>
<tr>
<td><code>a1 %a-% a2</code></td>
<td>Subtraction of arrays <code>a1</code> and <code>a2</code></td>
</tr>
<tr>
<td><code>a1 %a*% a2</code></td>
<td>Element-wise multiplication of arrays <code>a1</code> and <code>a2</code></td>
</tr>
<tr>
<td><code>a1 %a/% a2</code></td>
<td>Element-wise division of arrays <code>a1</code> and <code>a2</code></td>
</tr>
<tr>
<td><code>a1 %a/0% a2</code></td>
<td>Division of arrays <code>a1</code> and <code>a2</code> with division by zero</td>
</tr>
<tr>
<td><code>tab1 %a_% marg</code></td>
<td>Marginalization of <code>tab1</code> by <code>marg</code></td>
</tr>
<tr>
<td><code>tab1 %a==% tab2</code></td>
<td>Equality of <code>tab1</code> and <code>tab2</code></td>
</tr>
<tr>
<td><code>tab1 %a^% extra</code></td>
<td>Creation of dimensions <code>extra</code> for <code>tab1</code></td>
</tr>
<tr>
<td><code>tab1 %aperm% perm</code></td>
<td>Permutation of <code>perm</code> dimensions for <code>tab1</code></td>
</tr>
<tr>
<td><code>tab1 %aalign% tab2</code></td>
<td>Alignment of <code>tab1</code> and <code>tab2</code></td>
</tr>
<tr>
<td><code>tab1 %aslice% slice</code></td>
<td>Slicing <code>tab1</code> with <code>slice</code></td>
</tr>
<tr>
<td><code>tab1 %aslice*% slice</code></td>
<td>Slicing <code>tab1</code> with <code>slice</code></td>
</tr>
<tr>
<td><code>tab1 %amarg% marg</code></td>
<td>Marginalization of <code>tab1</code> by <code>marg</code></td>
</tr>
</tbody>
</table>

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tab1</code>, <code>tab2</code></td>
<td>Multidimensional arrays with named dimnames (we call them 'named arrays').</td>
</tr>
<tr>
<td><code>marg</code></td>
<td>A vector of indices or dimnames or a right hand sided formula giving the desired marginal.</td>
</tr>
<tr>
<td><code>extra</code></td>
<td>List defining the extra dimensions.</td>
</tr>
</tbody>
</table>
perm A vector of indices or dimnames or a right hand sided formula giving the desired
permuation.

slice A list of the form name=value.

a, a1, a2 Arrays (with named dimnames)

Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>

Examples
hec <- HairEyeColor
a1 <- tabMarg(hec, c("Hair", "Eye"))
a2 <- tabMarg(hec, c("Hair", "Sex"))
a3 <- tabMarg(hec, c("Eye", "Sex"))

## Binary operations
a1 %a+% a2
a1 %a-% a2
a1 %a*% a2
a1 %a/% a2

api-tabDist Marginalize and condition in multidimensional array.

Description
Marginalize and condition in a multidimensional array which is assumed to represent a discrete
multivariate distribution.

Usage
tabDist(tab, marg = NULL, cond = NULL, normalize = TRUE)

Arguments
tab Multidimensional array with dimnames.
marg A specification of the desired margin; a character vector, a numeric vector or a
right hand sided formula.
cond A specification of what is conditioned on. Can take two forms: Form one is a a
character vector, a numeric vector or a right hand sided formula. Form two is as
a simple slice of the array, which is a list of the form var1=value1, var2=value2
etc.
normalize Should the result be normalized to sum to 1.

Value
A multidimensional array.
Examples

hec <- HairEyeColor

is.named.array( hec )
## We need dimnames, and names on the dimnames

## Marginalize:
tabDist(hec, marg= ~Hair + Eye)
tabDist(hec, marg= ~Hair:Eye)
tabDist(hec, marg= c("Hair", "Eye"))
tabDist(hec, marg= 1:2)

## Condition
tabDist(hec, cond= ~Sex + Hair)
tabDist(hec, cond= ~Sex:Hair)
tabDist(hec, cond= c("Sex", "Hair"))
tabDist(hec, cond= c(3,1))

tabDist(hec, cond= list(Hair="Black"))
tabDist(hec, cond= list(Hair=1))

## Not run:
## This will fail
## tabDist(hec, cond= list(Hair=c("Black", "Brown")))
tabDist(hec, cond= list(Hair=1:2))

## End(Not run)
## But this will do the trick
a <- tabSlice(hec, slice=list(Hair=c("Black", "Brown")))
tabDist(a, cond=Hair)

## Combined
tabDist(hec, marg=~Hair+Eye, cond=~Sex)
tabDist(hec, marg=~Hair+Eye, cond="Sex")

tabDist(hec, marg=~Hair+Eye, cond=list(Sex="Male"))
tabDist(hec, marg=~Hair+Eye, cond=list(Sex="Male"), normalize=FALSE)

tabDist(hec, cond=list(Sex="Male"))
tabDist(hec, cond=list(Sex="Male"), normalize=FALSE)
Create multidimensional arrays

Description

Alternative ways of creating arrays

Usage

`tabNew(names, levels, values, normalize = "none", smooth = 0)`

Arguments

- `names`: Names of variables defining table; a character vector or a right hand sided formula.
- `levels`: 1. a list with specification of the levels of the factors in `names` or 2) a vector with number of levels of the factors in `names`. See 'examples' below.
- `values`: values to go into the parray
- `normalize`: Either "none", "first" or "all". Should result be normalized, see 'Details' below.
- `smooth`: Should values be smoothed, see 'Details' below.

Details

If `normalize="first"` then for each configuration of all other variables than the first, the probabilities are normalized to sum to one. Thus f(a,b,c) becomes a conditional probability table of the form p(a|b,c). If `normalize="all"` then the sum over all entries of f(a,b,c) is one.

If \code{smooth} is positive then \code{smooth} is added to \code{values} before normalization takes place.

Value

An array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
universe <- list(gender=c('male','female'),
                 answer=c('yes','no'),
                 rain=c('yes','no'))
t1 <- tabNew(c("gender","answer"), levels=universe, values=1:4)
t1
```
t2 <- tabNew(~gender:answer, levels=universe, values=1:4)
t2
t3 <- tabNew(~gender:answer, c(2,2), values=1:4)
t3

---

### Description

Interface functions and minor extensions to cpp functions.

### Usage

- `tabAdd(tab1, tab2)`
- `tabAlign(tab1, tab2)`
- `tabDiv(tab1, tab2)`
- `tabDiv0(tab1, tab2)`
- `tabOp(tab1, tab2, op = "+")`
- `tabEqual(tab1, tab2, eps = 1e-12)`
- `tabExpand(tab, aux, type = 0L)`
- `tabMult(tab1, tab2)`
- `tabSubt(tab1, tab2)`
- `tabListMult(lst)`
- `tabListAdd(lst)`
- `tabPerm(tab, perm)`
- `tabMarg(tab, marg = NULL)`
- `tabSum(tab, ...)`
- `tabProd(tab, ...)`
- `tabNormalize(tab, type = "none")`
Arguments

-op- The algebraic operation to be carried out.
-eps- Criterion for checking equality of two arrays.
-tab, tab1, tab2, ...- Arrays with named dimnames (we call them `named arrays`).
-aux- Either a list with names and dimnames or a named array from which such a list can be extracted.
-type- If 0 then entries are duplicated. If 3 then averages are computed. If 2 then 0 slices are inserted.
-lst- List of arrays.
-perm, marg- A vector of indices or dimnames or a right hand sided formula giving the desired permutation/margin.

Description

Table operations implemented in c++. Corresponding R functions without the trailing underscore exist.

Usage

- `tab_perm_(tab, perm)`
- `tab_expand_(tab, aux, type = 0L)`
- `tab_align_(tab1, tab2)`
- `tab_marg_(tab, marg)`
- `tab_op_(tab1, tab2, op = "*")`
- `tab_add_(tab1, tab2)`
- `tab_subt_(tab1, tab2)`
- `tab_mult_(tab1, tab2)`
- `tab_div_(tab1, tab2)`
- `tab_div0_(tab1, tab2)`
- `tab_equal_(tab1, tab2, eps = 1e-12)`
api_tabSlice

---

**Arguments**

- `tab`: Arrays with named dimnames (we call them 'named arrays').
- `perm`: A vector of indices or dimnames or a right hand sided formula giving the desired permutation/margin.
- `aux`: Either a list with names and dimnames or a named array from which such a list can be extracted.
- `type`: If 0 then entries are duplicated. If 3 then averages are computed. If 2 then 0 slices are inserted.
- `tab1`: Arrays with named dimnames (we call them 'named arrays').
- `tab2`: Arrays with named dimnames (we call them 'named arrays').
- `marg`: A vector of indices or dimnames or a right hand sided formula giving the desired permutation/margin.
- `op`: The operation to be carried out; "+", "-", "+", "/".
- `eps`: Criterion for checking equality of two arrays.
- `lst`: List of arrays.

---

**Description**

Functions for extracting slices of arrays

**Usage**

```r
api_tabSlice(tab, slice = NULL, margin = names(slice),
             drop = TRUE, as.array = FALSE)
```

```r
tabSlice2(tab, slice, margin.idx, drop = TRUE, as.array = FALSE)
```

```r
tabSlicePrim(tab, slice, drop = TRUE)
```

```r
tabSliceMult(tab, slice, val = 1, comp = 0)
```

```r
tabSlice2Entries(tab, slice, complement = FALSE)
```
Simulate data (slice of) an array: Simulate n observations from the array x conditional on the variables in margin (a vector of indices) takes values given by margin.value.

**Arguments**

- **tab**: An array with named dimnames.
- **slice**: A list defining the slice.
- **margin**: Names of variables in slice.
- **drop**: If TRUE then dimensions with only one level will be dropped from the output.
- **as.array**: If the resulting array is one-dimensional the result will by default be a vector with no dim attribute unless as.array is TRUE.
- **margin.idx**: Indec of variables in slice.
- **val**: The values that entries in the slice will be multiplied with.
- **comp**: The values that entries NOT in the slice will be multiplied with.
- **complement**: If TRUE the complement of the entries are returned.

**Examples**

```r
x = HairEyeColor
s = list(Hair=c("Black", "Brown"), Eye=c("Brown", "Blue"))

s1 = tabSlice(x, slice=s); s1

tabSlice2Entries(x, slice=s)

## tabSliceMult
s2 = tabSliceMult(x, slice=s); s2

sp = list(c(1,2), c(1,2), TRUE)

tabSlicePrim(x, slice=sp)

# tabSlice(x, slice=s)
```

---

**array-simulate**

Simulate data from array.

**Description**

Simulate data (slice of) an array: Simulate n observations from the array x conditional on the variables in margin (a vector of indices) takes values given by margin.value.

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>
Usage

simulateArray(x, nsim = 1, margin, value.margin, seed = NULL)

## S3 method for class 'table'
simulate(object, nsim = 1, seed = NULL, margin, value.margin, ...)

## S3 method for class 'xtabs'
simulate(object, nsim = 1, seed = NULL, margin, value.margin, ...)

## S3 method for class 'array'
simulate(object, nsim = 1, seed = NULL, margin, value.margin, ...)

Arguments

x, object An array.
nsim Number of cases to simulate.
margin, value.margin
  Specification of slice of array to simulate from.
seed Seed to be used for random number generation.
... Additional arguments, currently not used.

Value

A matrix.

Note

The current implementation is fragile in the sense that it is not checked that the input argument x is an array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

## 2x2 array
x <- parray(c(“a”, “b”), levels=c(2, 2), values=1:4)

## Simulate from entire array
s <- simulateArray(x, 1000)
xtabs(~., as.data.frame(s))

## Simulate from slice defined by that dimension 1 is fixed at level 2
s <- simulateArray(x, 6000, 1, 2)
xtabs(~., as.data.frame(s))

## 2 x 2 x 2 array
x <- parray(c(“a”, “b”, “c”), levels=c(2, 2, 2), values=1:8)
## Simulate from entire array
s <- simulateArray(x, 36000)
xtabs(~., as.data.frame(s))

## Simulate from slice defined by that dimension 3 is fixed at level 1
s <- simulateArray(x, 10000, 3, 1)
xtabs(~., as.data.frame(s))
dag2chol

Usage

cov2pcor(V)
conc2pcor(K)

Arguments

V       Covariance matrix
K       Concentration matrix

Value

A matrix with the same dimension as V.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

data(math)
S <- cov.wt(math)$cov
cov2pcor(S)

dag2chol     Create regression matrix matrix from DAG A DAG can be represented as a triangular matrix of regression coefficients.

Description

Create regression matrix matrix from DAG A DAG can be represented as a triangular matrix of regression coefficients.

Usage

dag2chol(object)

Arguments

object       A graph, either a graphNEL or an igraph object.
Examples

g <- dag(~x2|x1 + x3|x1:x2 + x4|x3)
gi <- as(g, "igraph")
dag2chol(g)
dag2chol(gi)

Description

This dataset comes from a study of symptoms of crown dieback, cankers and symptoms caused by other pathogens and pests in ash trees (Fraxinus excelsior). In all 454 trees were observed in two plots. There are 8 categorical variables, 6 of which are binary and two are trichotomous with values representing increasing severity of symptoms, and one continuous variable, tree diameter at breast height (DBH).

Usage

data(ashtrees)

Format

A data frame with 454 observations on the following 9 variables.

plot  a factor with levels 2 6
dieback a factor with levels 0 1 2
dead50  a factor with levels 0 0.5 1
bushy  a factor with levels 0 1
canker a factor with levels BRNCH MAIN NONE
wilt   a factor with levels 0 1
roses  a factor with levels 0 1
discolour a factor with levels 0 1
dbh    a numeric vector

References


Examples

data(ashtrees)
head(ashtrees)
data-BodyFat

**Body Fat Data**

---

**Description**

Estimates of the percentage of body fat determined by underwater weighing and various body circumference measurements for 252 men.

**Usage**

data(BodyFat)

data(BodyFat)

**Format**

A data frame with 252 observations on the following 15 variables.

- **Density**  Density determined from underwater weighing, a numeric vector
- **BodyFat**  Percent body fat from Siri’s (1956) equation, a numeric vector
- **Age**  in years, a numeric vector
- **Weight**  in lbs, a numeric vector
- **Height**  in inches, a numeric vector
- **Neck**  circumference in cm, a numeric vector
- **Chest**  circumference in cm, a numeric vector
- **Abdomen**  circumference in cm, a numeric vector
- **Hip**  circumference in cm, a numeric vector
- **Thigh**  circumference in cm, a numeric vector
- **Knee**  circumference in cm, a numeric vector
- **Ankle**  circumference in cm, a numeric vector
- **Biceps**  circumference in cm, a numeric vector
- **Forearm**  circumference in cm, a numeric vector
- **Wrist**  circumference in cm, a numeric vector

**Source**

For more information see https://lib.stat.cmu.edu/datasets/bodyfat
data-breastcancer

References


Examples

data(BodyFat)
head(BodyFat)

data-breastcancer

Description

Perturbations of the p53 pathway are associated with more aggressive and therapeutically refractory tumours. We preprocessed the data using Robust Multichip Analysis (RMA). Dataset has been truncated to the 1000 most informative genes (as selected by Wilcoxon test statistics) to simplify computation. The genes have been standardised to have zero mean and unit variance (i.e. z-scored).

Usage

data(breastcancer)

Format

A data frame with 250 observations on 1001 variables. The first 1000 columns are numerical variables; the last column (named code) is a factor with levels case and control.

Details

The factor code defines whether there was a mutation in the p53 sequence (code=case) or not (code=control).

Source

Dr. Chris Holmes, c.holmes at stats dot. ox . ac .uk
References


Examples

```r
data(breastcancer)
## maybe str(breastcancer) ; plot(breastcancer) ...
```

---

**data-cad**

*Coronary artery disease data*

**Description**

A cross classified table with observational data from a Danish heart clinic. The response variable is CAD (coronary artery disease, sometimes called heart attack).

**Usage**

```r
data(cad1)
```

**Format**

A data frame with 236 observations on the following 14 variables.

- **Sex**  Sex; a factor with levels Female Male
- **AngPec**  Angina pectoris (chest pain attacks); a factor with levels Atypical None Typical
- **AMI**  Acute myocardic infarct; a factor with levels Definite NotCertain
- **QWave**  A reading from an electrocardiogram; a factor with levels No Yes; Yes means pathological and is a sign of previous myocardial infarction.
- **QWavecode**  a factor with levels Nonusable Usable. An assessment of whether QWave is reliable.
- **STCode**  a factor with levels Nonusable Usable. An assessment of whether STchange is reliable.
- **STchange**  A reading from an electrocardiogram; a factor with levels No Yes. An STchange indicates a blockage of the coronary artery.
- **SuffHeartF**  Sufficient heart frequency; a factor with levels No Yes
- **Hypertrophi**  a factor with levels No, Yes. Hypertrophy refers to an increased size of the heart muscle due to exercise.
- **Hyperchol**  a factor with levels No Yes. Hypercholesterolemia, also called high cholesterol, is the presence of high levels of cholesterol in the blood.
- **Smoker**  Is the patient a smoker; a factor with levels No, Yes.
- **Inherit**  Hereditary predispositions for CAD; a factor with levels No, Yes.
- **Heartfail**  Previous heart failures; a factor with levels No Yes
- **CAD**  Coronary Artery Disease; a factor with levels No Yes. CAD refers to a reduction of blood flow to the heart muscle (commonly known as a heart attack). The diagnosis made from biopsies.
Details

Notice that data are collected at a heart clinic, so data do not represent the population, but are conditional on patients having ended up at the clinic.

- cad1: Complete dataset, 236 cases.
- cad2: Incomplete dataset, 67 cases. Information on (some of) the variables 'Hyperchol', 'Smoker' and 'Inherit' is missing.

References


Examples

```r
data(cad1)
## maybe str(cad1) ; plot(cad1) ...
```

---

data-carcass

Lean meat contents of 344 pig carcasses

Description

Measurement of lean meat percentage of 344 pig carcasses together with auxiliary information collected at three Danish slaughter houses

Usage

data(carcass)

Format

carcassall: A data frame with 344 observations on the following 17 variables.

- weight  Weight of carcass
- lengthc  Length of carcass from back toe to head (when the carcass hangs in the back legs)
- lengthf  Length of carcass from back toe to front leg (that is, to the shoulder)
- lengthp  Length of carcass from back toe to the pelvic bone
- Fat02, Fat03, Fat11, Fat12, Fat13, Fat14, Fat16  Thickness of fat layer at different locations on the back of the carcass (FatXX refers to thickness at (or rather next to) rib no. XX. Notice that 02 is closest to the head
Meat11, Meat12, Meat13  Thickness of meat layer at different locations on the back of the carcass, see description above
LeanMeat  Lean meat percentage determined by dissection
slaughterhouse  Slaughter house; a factor with levels a b c
sex  Sex of the pig; a factor with a b c. Notice that it is no an error to have three levels; the third level refers to castrates

Note
carcass: Contains only the variables Fat11, Fat12, Fat13, Meat11, Meat12, Meat13, LeanMeat

Source

Examples
data(carcass)
head(carcass)

data-chestSim  Simulated data from the Chest Clinic example

Description
Simulated data from the Chest Clinic example (also known as the Asia example) from Lauritzen and Spiegelhalter, 1988 (see reference below).

Usage
data(chestSim500)

Format
A data frame with 500 observations on the following 8 variables.
asia  Recent visit to Asia?; a factor with levels yes no
tub  Has tuberculosis?; a factor with levels yes no
smoke  Is a smoker?; a factor with levels yes no
lung  Has lung cancer?; a factor with levels yes no
bronc  Has bronchitis?; a factor with levels yes no
either  Either lung cancer or tuberculosis?; a factor with levels yes no
xray  Positive x-ray? a factor with levels yes no
dysp  Dyspnoea (shortness of breath)?; a factor with levels yes no
Details

Notice that the chest clinic example is a contrived example; it does not originate from an empirical study.

References


Examples

data(chestSim500)
### maybe str(chestSim500) ; plot(chestSim500) ...

data(dietox)

Description

The dietox data frame has 861 rows and 7 columns.

Usage

data(dietox)

Format

This data frame contains the following columns: Weight, Feed, Time, Pig, Evit, Cu, Litter.

Source


Examples

data(dietox)
**data-dumping**

**Gastric Dumping**

**Description**

A contingency table relating surgical operation, centre and severity of gastric dumping, a syndrome associated with gastric surgery.

**Usage**

data(dumping)

**Format**

A 3x4x4 table of counts cross-classified by Symptom (none/slight/moderate), Operation (Vd/Va/Vh/Gr) and Centre (1:4).

**Details**

Gastric dumping syndrome is a condition where ingested foods bypass the stomach too rapidly and enter the small intestine largely undigested. It is an undesirable side-effect of gastric surgery. The table summarizes the results of a study comparing four different surgical operations on patients with duodenal ulcer, carried out in four centres, as described in Grizzle et al (1969). The four operations were: vagotomy and drainage, vagotomy and antrectomy (removal of 25\% removal of 50\% removal of 75\%).

**Source**


**Examples**

data(dumping)
plot(dumping)

**data-lizard**

**Lizard behaviour**

**Description**

In a study of lizard behaviour, characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H). Perch means preferred place to settle down (a branch on a tree). The focus of interest is in how the propensities of the lizards to choose perch height and diameter are related, and whether and how these depend on species.
Usage

data(lizard)

Format

A 3-dimensional array with factors diam: "<=4" ">4" height: ">4.75" "<=4.75" species: "anoli" "dist"

References


Examples

data(lizard)

# Datasets lizardRAW and lizardDF are generated with the following code
lizardAGG <- as.data.frame(lizard)
f <- lizardAGG$Freq
idx <- unlist(mapply(function(i, n) rep(i, n), 1:8, f))
set.seed(0805)
idx <- sample(idx)
lizardRAW <- as.data.frame(lizardAGG[idx, 1:3])
rownames(lizardRAW) <- 1:NROW(lizardRAW)

---

data-mathmark  Mathematics marks for students

Description

The mathmark data frame has 88 rows and 5 columns.

Usage

data(mathmark)

Format

This data frame contains the following columns: mechanics, vectors, algebra, analysis, statistics.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>
References


Examples

data(mathmark)

---

data-mildew  Mildew fungus

Description

The data stem from a cross between two isolates of the barley powdery mildew fungus. For each offspring 6 binary characteristics, each corresponding to a single locus, were recorded. The object of the analysis is to determine the order of the loci along the chromosome.

Usage

data(mildew)

Format

A 6 dimensional array where each variable has levels "1" and "2". The variables are: la10, locc, mp58, c365, p53a and a365.

References


Examples

data(mildew)
## maybe str(mildew) ; plot(mildew) ...
Description

Data from an experiment on composition of sow milk. Milk composition is measured on four occasions during lactation on a number of sows. The treatments are different types of fat added to the sows feed.

Usage

data(milkcomp)

Format

A data frame with 214 observations on the following 7 variables.

- `sow` a numeric vector
- `lactime` a numeric vector
- `treat` a factor with levels `a b c d e f g`
- `fat` a numeric vector
- `protein` a numeric vector
- `dm` (dry matter) a numeric vector
- `lactose` a numeric vector

Details

- `a` is the control, i.e. no fat has been added.
- `fat + protein + lactose` almost add up to `dm` (dry matter)

References


Examples

data(milkcomp)
### maybe str(milk) ; plot(milk) ...
Description

The data come from a study of the effects of five dietary regimens with different fatty acid compositions on liver lipids and hepatic gene expression in 40 mice.

Usage

data(Nutrimouse)

Format

A data frame with 40 observations on 143 variables of which two are factors and 141 are numeric.

- genotype: a factor with levels wt ppar
- diet: a factor with levels coc fish lin ref sun

Details

The data come from a study of the effects of five dietary regimens with different fatty acid compositions on liver lipids and hepatic gene expression in wild-type and PPAR-alpha-deficient mice (Martin et al., 2007).

There were 5 replicates per genotype and diet combination.

There are two design variables: (i) genotype, a factor with two levels: wild-type (wt) and PPAR-alpha-deficient (ppar), and (ii) diet, a factor with five levels. The oils used for experimental diet preparation were: corn and colza oils (50/50) for a reference diet (ref); hydrogenated coconut oil for a saturated fatty acid diet (coc); sunflower oil for an Omega6 fatty acid-rich diet (sun); linseed oil for an Omega3-rich diet (lin); and corn/colza/enriched (43/43/14) fish oils (fish).

There are 141 response variables: (i) the log-expression levels of 120 genes measured in liver cells, and (ii) the concentrations (in percentages) of 21 hepatic fatty acids measured by gas chromatography.

Source

The data were provided by Pascal Martin from the Toxicology and Pharmacology Laboratory, National Institute for Agronomic Research, France.

References

Examples

```r
data(Nutrimouse)
```

---

**data-personality**  
*Personality traits*

---

**Description**

The personality dataframe has 240 rows and 32 columns

**Usage**

```r
data(personality)
```

**Format**

This dataframe has recordings on the following 32 variables: distant, talkatv, carelss, hardwrk, anxious, agreebl, tense, kind, opposng, relaxed, disorgn, outgoin, approvn, shy, discipl, harsh, persever, friendl, worryin, respnsi, contrar, sociabl, lazy, coopera, quiet, organiz, criticl, lax, laidback, withdrw, givinup, easygon

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**References**

Origin unclear

**Examples**

```r
data(personality)
str(personality)
```
**data-rats**

**Weightloss of rats**

**Description**
An artificial dataset. 24 rats (12 female, 12 male) have been randomized to use one of three drugs (products for loosing weight). The weightloss for each rat is noted after one and two weeks.

**Usage**
data(rats)

**Format**
A dataframe with 4 variables. Sex: "M" (male), "F" (female). Drug: "D1", "D2", "D3" (three types). W1 weightloss, week one. W2 weightloss, week 2.

**References**

---

**data-reinis**

**Risk factors for coronary heart disease.**

**Description**
Data collected at the beginning of a 15 year follow-up study of probable risk factors for coronary thrombosis. Data are from all men employed in a car factory.

**Usage**
data(reinis)

**Format**

**References**
Description

Using chemical analysis determine the origin of wines

Usage

data(wine)

Format

A data frame with 178 observations on the following 14 variables.

- Cult: a factor with levels v1, v2, v3: 3 different graph varieties
- Alch: Alcohol
- Mlca: Malic acid
- Ash: Ash
- Aloa: Alcalinity of ash
- Mgns: Magnesium
- Ttlp: Total phenols
- Flvn: Flavanoids
- Nnfp: Nonflavanoid phenols
- Prnt: Proanthocyanins
- Cli: Color intensity
- Hue: Hue
- Oodw: OD280/OD315 of diluted wines
- Prln: Proline

Details

Data comes from the UCI Machine Learning Repository. The grape variety Cult is the class identifier.

Source


References

See references at https://archive.ics.uci.edu/ml/datasets/Wine/
Examples

data(wine)
## maybe str(wine) ; plot(wine) ...

Description

Downstream aliases for other graphical modelling packages. Will be deprecated in due course.

fastcombn

Generate All Combinations of n Elements Taken m at a Time

Description

Generate all combinations of the elements of x taken m at a time. If x is a positive integer, returns all combinations of the elements of seq(x) taken m at a time.

Usage

fastcombn(x, m, FUN = NULL, simplify = TRUE, ...)

combn_prim(x, m, simplify = TRUE)

Arguments

x vector source for combinations, or integer n for x <- seq(n).

m number of elements to choose.

FUN function to be applied to each combination; default ‘NULL’ means the identity, i.e., to return the combination (vector of length ‘m’).

simplify logical indicating if the result should be simplified to a matrix; if FALSE, the function returns a list.

... Further arguments passed on to FUN.

Details

• Factors x are accepted.
• combn_prim is a simplified (but faster) version of the combn function. Does nok take the FUN argument.
• fastcombn is intended to be a faster version of the combn function.
Value

A matrix or a list.

Author(s)

Søren Højsgaard

See Also

combn

Examples

```r
x <- letters[1:5]; m <- 3
fastcombn(x, m)
combn(x, m)
combn_prim(x, m)

x <- letters[1:4]; m <- 3
fastcombn(x, m, simplify=FALSE)
combn(x, m, simplify=FALSE)
combn_prim(x, m, simplify=FALSE)

x <- 1:10; m <- 3
fastcombn(x, m, min)
combn(x, m, min)

x <- factor(letters[1:8]); m <- 5

if (require(microbenchmark)){
  microbenchmark(
    combn(x, m, simplify=FALSE),
    combn_prim(x, m, simplify=FALSE),
    fastcombn(x, m, simplify=FALSE),
    times=50
  )
}
```
Usage

as.adjMAT(object, result = "matrix")

Arguments

object: An object to be coerced.
result: The format to be coerced to.

Description

Return a list of (maximal) cliques of an undirected graph.

Usage

gc <- get_cliques(G)

Arguments

object: An undirected graph represented either as a graphNEL object, an igraph object, a (dense) matrix, a (sparse) dgCMatrix
amat: An adjacency matrix.

Details

In graph theory, a clique is often a complete subset of a graph. A maximal clique is a clique which can not be enlarged. In statistics (and that is the convention we follow here) a clique is usually understood to be a maximal clique.

Finding the cliques of a general graph is an NP complete problem. Finding the cliques of triangulated graph is linear in the number of cliques.

The workhorse is the max_cliqueMAT function which calls the maxClique function in the RBGL package.

Value

A list.
Synonymous functions

For backward compatibility with downstream packages we have the following synonymous functions:

- `getCliques = get_cliques`
- `maxCliqueMAT = max_cliqueMAT`

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`ug`, `dag`, `mcs`, `mcsMAT`, `rip`, `ripMAT`, `moralize`, `moralizeMAT`

Examples

```r
## graphNEL
uG0 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a) # a graphNEL object
get_cliques(uG0)

uG1 <- as(uG0, "igraph")
get_cliques(uG1)

uG2 <- as(uG0, "matrix")
get_cliques(uG2)

uG3 <- as(uG1, "dgCMatrix")
get_cliques(uG3)
```

---

**Description**

Methods for changing graph representations

**Usage**

```r
coerceGraph(object, class)

graph_as(object, outtype, intype = NULL)
```

**Arguments**

- `object` A graph object
- `class` The desired output class
- `outtype` The desired output outtype
- `intype` The desired output outtype (only relevant if object is a list)
Details

coeffeGraph is used in the book "Graphical models with R". A more generic approach is as().

Examples

g1 <- ug(~a:b+b:c)
as(g1, "igraph")
as(g1, "matrix")
as(g1, "Matrix")
as(g1, "dgCMatrix")

## graph_as(g1, "ugList") ## Fails
## getClques(g1) ## Works
l1 <- list(c("a", "b"), c("b", "c"))
graph_as(l1, "graphNEL", "ugList")
g_ig2dm_(object)
g_ig2sm_(object)
g_xm2gn_(object)
g_xm2ig_(object)
g_xm2dm_(object)
g_xm2sm_(object)
g_xm2xm_(object, result = "matrix")
g_gn2xm_(object, result = "matrix")
g_gn2ftM_(object)
g_gn2tfM_(object)
graphNEL2adjMAT(object, result = "matrix")

Arguments

object An object representing a graph
result Either 'matrix' (dense) or 'dgCMatrix' (sparse, can be abbreviated to 'Matrix').

Details
No checking is made. In the function the following names are used:

- "ig": "igraph";
- "gn": "graphNEL";
- "sm": "dgCMatrix" (sparse matrix);
- "dm": "matrix" (dense matrix)

Synonymous functions
For backward compatibility with downstream packages we have the following synonymous functions:

- graphNEL2adjMAT = g_gn2xm_ (Used in HydeNet)
- graphNEL2M = g_gn2xm_ (Used in simPATHy)
- M2graphNEL = g_xm2gn_ (Used in simPATHy)

Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>
See Also

ug, dag

---

Description
Coercion of graphs represented as lists to various graph formats.

Usage

```r
g_ugl2gn_(glist, vn = NULL)
g_ugl2ig_(zz, vn = NULL)
g_ugl2dm_(zz, vn = NULL)
g_ugl2sm_(zz, vn = NULL)
g_ugl2XX_(zz, outtype, vn = NULL)
g_dagl2gn_(glist, vn = NULL)
g_dagl2ig_(zz, vn = NULL)
g_dagl2dm_(zz, vn = NULL)
g_dagl2sm_(zz, vn = NULL)
g_dagl2XX_(zz, outtype, vn = NULL)
g_adl2gn_(zz)
g_adl2ig_(zz)
g_adl2dm_(zz)
g_adl2sm_(zz)
g_adl2XX_(zz, outtype)
g_M2adl_(amat)
g_M2ugl_(amat)
```
g_M2dagl_(amat)

g_ugl2M_(glist, vn = NULL, result = "matrix")

g_dagl2M_(glist, vn = NULL, result = "matrix")

g_adl2M_(alist, result = "matrix")

**Arguments**

- **glist**: A list of generators where a generator is a character vector. If interpreted as generators of an undirected graph, a generator is a complete set of vertices in the graph. If interpreted as generators of a dag, a generator \((v_1, \ldots, v_n)\) means that there will be arrows from \(v_2, \ldots, v_n\) to \(v_1\).
- **vn**: The names of the vertices in the graphs. These will be the row and column names of the matrix.
- **zz**: An object representing a graph.
- **outtype**: What should a list be coerced to.
- **amat**: Adjacency matrix (dense or sparse dgCMatrix).
- **result**: A graph object.
- **alist**: An adjacency list.

**Examples**

```r
## Sparse and dense adjacency matrices converted to adjacency list
g1 <- ug(~a:b + b:c + c:d, result="matrix")
g2 <- ug(~a:b + b:c + c:d, result="dgCMatrix")
g_M2adl_( g1 )
## Sparse and dense adjacency matrices converted to cliques
g_M2ugl_( g1 )
## Sparse and dense adjacency matrices converted to cliques
g_M2dagl_( g1 )
## g_M2adl_( g2 ) ## FIXME FAILS for sparse matrix
## g_M2ugl_( g2 ) ## FIXME Is there an issue here??
## g_M2dagList( g2 ) ## Fails for sparse matrix
```

**Description**

These functions are wrappers for creation of graphs as implemented by graphNEL objects in the graph package.
Usage

ug(..., result = "graphNEL")

ugi(...)

ugList(x, result = "graphNEL")

dag(..., result = "graphNEL", forceCheck = FALSE)

dagi(..., forceCheck = FALSE)

dagList(x, result = "graphNEL", forceCheck = FALSE)

Arguments

... A generating class for a graph, see examples below
result The format of the graph. The possible choices are "graphNEL" (for a graphNEL object), "igraph" (for an igraph object), "matrix" (for an adjacency matrix), "dgCMatrix" (for a sparse matrix).
x A list or individual components from which a graph can be created.
forceCheck Logical determining if it should be checked if the graph is acyclical. Yes, one can specify graphs with cycles using the dag() function.

Value

Functions ug(), and dag() can return a graphNEL object, an igraph object, a sparse or a dense adjacency matrix.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

## The following specifications of undirected graphs are equivalent:
uG1 <- ug(~ a:b:c + c:d)
uG2 <- ug(c("a", "b", "c"), c("c", "d"))
uG3 <- ug(c("a", "b"), c("a", "c"), c("b", "c"), c("c", "d"))

graph::edges(uG1)
graph::nodes(uG1)

## The following specifications of directed acyclic graphs are equivalent:
daG1 <- dag(~ a:b:c + b:c + c:d)
daG2 <- dag(c("a", "b", "c"), c("b", "c"), c("c", "d"))

graph::edges(daG1)
graph::nodes(daG2)
## dag() allows to specify directed graphs with cycles:

dag4 <- dag(~ a:b + b:c + c:a) # A directed graph but with cycles

## A check for acyclicity can be done with
## dag5 <- dag(~ a:b + b:c + c:a, forceCheck=TRUE)

## A check for acyclicity is provided by topoSort
topo_sort( dag2 )
topo_sort( dag4 )

## Different representations
uG6 <- ug(~a:b:c + c:d, result="graphNEL") # default
uG7 <- ug(~a:b:c + c:d, result="igraph") # igraph
uG8 <- ug(~a:b:c + c:d, result="matrix") # dense matrix
uG9 <- ug(~a:b:c + c:d, result="dgCMatrix") # sparse matrix

---

**graph-edgeList**

*Find edges in a graph and edges not in a graph.*

### Description

Returns the edges of a graph (or edges not in a graph) where the graph can be either a graphNEL object, an igraph object or an adjacency matrix.

### Usage

- `edgeList(object, matrix = FALSE)`
- `edgeListMAT(adjmat, matrix = FALSE)`
- `nonEdgeList(object, matrix = FALSE)`
- `nonEdgeListMAT(adjmat, matrix = FALSE)`

### Arguments

- **object**: A graphNEL object, an igraph object, a dense matrix or a sparse dgCMatrix (the two latter representing an adjacency matrix).
- **matrix**: If TRUE the result is a matrix; otherwise the result is a list.
- **adjmat**: An adjacency matrix.

### Examples

## A graph with edges

g <- ug(~a:b + b:c + c:d)
gm <- as(g, "matrix")
edgelist(g)
Properties of a generating class (for defining a graph).

Description

A set of generators define an undirected graph, here called a dependence graph. Given a set of generators it is checked 1) if the dependence dependence graph is in 1-1-correspondance with the genrators (such that the corresponding model is graphical) and 2) if the dependence graph is chordal (triangulated) (such that the corresponding model is decomposable).

Usage

isGraphical(x)

isDecomposable(x)

Arguments

x

A generating class given as right hand sided formula or a list; see examples below.

Details

A set of sets of variables, say A_1, A_2, ... A_K is called a generating class for a graph with vertices V and edges E. If two variables a,b are in the same generator, say A_j, then a and b are vertices in the graph and there is an undirected edge between a and b.
The graph induced by \code{\texttt{g1 = \-a:b + a:c + b:c + c:d}} has edges \code{ab, ac, bc, cd}. The cliques of this graph are \code{abc, cd}. Hence there is not a 1-1-correspondance between the graph and the generators.

On the other hand, \code{\texttt{g2 <- \-a:b:c + c:d}} induces the same graph in this case there is a 1-1-correspondance.

The graph induced by \code{\texttt{g3 <- \-a:b + b:c + c:d + d:a}} is in 1-1-correspondance with its dependence graph, but the graph is not chordal.

**Value**

TRUE or FALSE

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**See Also**

\code{mcs}, \code{rip}

**Examples**

```r
\begin{verbatim}
g1 <- \-a:b + a:c + b:c + c:d
g2 <- \-a:b:c + c:d
g3 <- \-a:b + b:c + c:d + d:a

isGraphical( g1 ) # FALSE
isGraphical( g2 ) # TRUE
isGraphical( g3 ) # TRUE

isDecomposable( g1 ) # FALSE
isDecomposable( g2 ) # TRUE
isDecomposable( g3 ) # TRUE

## A generating class can be given as a list:
f <- list(c("a","b"), c("b","c"), c("a","c"))
isGraphical( f )
isDecomposable( f )
\end{verbatim}
```
**graph-iplot**

*Function for plotting graphs using the 'igraph' package.*

**Description**

Generic function for plotting graphs using the 'igraph' package and a plot method for graphNEL objects.

**Usage**

```r
iplot(x, ...)
```

## S3 method for class 'graphNEL'

```r
iplot(x, ...)
```

**Arguments**

- `x` A graph object to be plotted.
- `...` Additional arguments

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**Examples**

```r
UG <- ug(~a:b+b:c:d)
iplot(UG)
```

---

**graph-is**

*Check properties of graphs.*

**Description**

Check if a graph is 1) a directed acyclic graph (DAG), 2) a directed graph (DG), 3) an undirected graph (UG), 4) a triangulated (chordal) undirected graph (TUG).
Usage

is_dag(object)

is_dagMAT(object)

is_ug(object)

is_ugMAT(object)

is_tug(object)

is_tugMAT(object)

is_dg(object)

is_dgMAT(object)

is_adjMAT(object)

is.adjMAT(object)

Arguments

object A graph represented as a graphNEL (graph package), an igraph (igraph package), an adjacency matrix or a sparse adjacency matrix (a dgCMatrix from the Matrix package).

Details

- A non-zero value at entry (i,j) in an adjacency matrix A for a graph means that there is an edge from i to j. If also (j,i) is non-zero there is also an edge from j to i. In this case we may think of a bidirected edge between i and j or we may think of the edge as being undirected. We do not distinguish between undirected and bidirected edges in the gRbase package. On the other hand, graphNEL objects from the graph package makes such a distinction (the function edgemode() will tell if edges are "directed" or "undirected" in a graphNEL object).
- The function is_ug() checks if the adjacency matrix is symmetric (If applied to a graphNEL, the adjacency matrix is created and checked for symmetry.)
- The function is_tug() checks if the graph is undirected and triangulated (also called chordal) by checking if the adjacency matrix is symmetric and the vertices can be given a perfect ordering using maximum cardinality search.
- The function is_dg() checks if a graph is directed, i.e., that there are no undirected edges. This is done by computing the elementwise product of A and the transpose of A; if there are no non-zero entries in this product then the graph is directed.
- The function is_dag() will return TRUE if all edges are directed and if there are no cycles in the graph. (This is checked by checking if the vertices in the graph can be given a topological ordering which is based on identifying an undirected edge with a bidirected edge).
• There is a special case, namely if the graph has no edges at all (such that the adjacency matrix consists only of zeros). Such a graph is both undirected, triangulated, directed and directed acyclic.

**Synonymous functions**

The functions

- `is.TUG/is.DAG/is.DG/is.UG/is.adjMAT`

are synonymous with

- `is_tug/is_dag/is dg/is_ug/is_adjMAT`.

The `is.X` group of functions will be deprecated.

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**See Also**

`dag, ug`

**Examples**

```r
## DAGs
dagNEL <- dag(~ a:b + c:d + e, result="graphNEL")

## Undirected graphs
ugNEL <- ug(~ a:b + c:d + e, result="graphNEL")

## Is graph a DAG?
is_dag(dagNEL)
is_dag(ugNEL)

## Is graph an undirected graph
is_ug(dagNEL)
is_ug(ugNEL)

## Is graph a triangulated (i.e. chordal) undirected graph
is_tug(dagNEL)
is_tug(ugNEL)

## Example where the graph is not triangulated
ug2NEL <- ug(~ a:b + b:c + c:d + d:a, result="graphNEL")
is_tug(ug2NEL)

## Bidirected graphs
graph::edgemode(ugNEL)
graph::edgemode(ugNEL) <- "directed"
graph::edgemode(ugNEL)
```
is_dag(ugNEL)
is_ug(ugNEL)

graph-mcs

Maximum cardinality search on undirected graph.

Description

Returns (if it exists) a perfect ordering of the vertices in an undirected graph.

Usage

mcs(object, root = NULL, index = FALSE)

## Default S3 method:
mcs(object, root = NULL, index = FALSE)
mcsMAT(amat, vn = colnames(amat), root = NULL, index = FALSE)
mcs_marked(object, discrete = NULL, index = FALSE)

## Default S3 method:
mcs_marked(object, discrete = NULL, index = FALSE)
mcs_markedMAT(amat, vn = colnames(amat), discrete = NULL, index = FALSE)

Arguments

object An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
root A vector of variables. The first variable in the perfect ordering will be the first variable on ‘root’. The ordering of the variables given in ‘root’ will be followed as far as possible.
index If TRUE, then a permutation is returned
amat Adjacency matrix
vn Nodes in the graph given by adjacency matrix
discrete A vector indicating which of the nodes are discrete. See ’details’ for more information.

Details

An undirected graph is decomposable iff there exists a perfect ordering of the vertices. The maximum cardinality search algorithm returns a perfect ordering of the vertices if it exists and hence this algorithm provides a check for decomposability. The mcs() functions finds such an ordering if it exists.
The notion of strong decomposability is used in connection with e.g. mixed interaction models where some vertices represent discrete variables and some represent continuous variables. Such graphs are said to be marked. The \code{mcsmarked()} function will return a perfect ordering iff the graph is strongly decomposable. As graphs do not know about whether vertices represent discrete or continuous variables, this information is supplied in the \code{discrete} argument.

Value

A vector with a linear ordering (obtained by maximum cardinality search) of the variables or character(0) if such an ordering can not be created.

Note

The workhorse is the \code{mcsMAT} function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

\code{moralize}, \code{junction_tree}, \code{rip}, \code{ug}, \code{dag}

Examples

mcs(uG)
mcsMAT(as(uG, "matrix"))
## Same as
uG <- ug(~ me:ve + me:al + ve:al + al:an + al:st + an:st, result="matrix")
mcsMAT(uG)

## Marked graphs
uG1 <- ug(~ a:b + b:c + c:d)
uG2 <- ug(~ a:b + a:d + c:d)
## Not strongly decomposable:
mcs_marked(uG1, discrete=c("a","d"))
## Strongly decomposable:
mcs_marked(uG2, discrete=c("a","d"))
graph-min-triangulate

**Minimal triangulation of an undirected graph**

### Description

An undirected graph $uG$ is triangulated (or chordal) if it has no cycles of length $\geq 4$ without a chord which is equivalent to that the vertices can be given a perfect ordering. Any undirected graph can be triangulated by adding edges to the graph, so called fill-ins which gives the graph $TuG$. A triangulation $TuG$ is minimal if no fill-ins can be removed without breaking the property that $TuG$ is triangulated.

### Usage

```r
minimal_triang(
  object,
  tobject = triangulate(object),
  result = NULL,
  details = 0
)
```

`minimal_triangMAT(amat, tamat = triangulateMAT(amat), details = 0)`

### Arguments

- **object**: An undirected graph represented either as a graphNEL object, a (dense) matrix, a (sparse) dgCMatrix.
- **tobject**: Any triangulation of object; must be of the same representation.
- **result**: The type (representation) of the result. Possible values are "graphNEL", "matrix", "dgCMatrix". Default is the same as the type of object.
- **details**: The amount of details to be printed.
- **amat**: The undirected graph which is to be triangulated; a symmetric adjacency matrix.
- **tamat**: Any triangulation of object; a symmetric adjacency matrix.

### Details

For a given triangulation $tobject$ it may be so that some of the fill-ins are superfluous in the sense that they can be removed from $tobject$ without breaking the property that $tobject$ is triangulated. The graph obtained by doing so is a minimal triangulation.

**Notice**: A related concept is the minimum triangulation, which is the graph with the smallest number of fill-ins. The minimum triangulation is unique. Finding the minimum triangulation is NP-hard.
Moralize a directed acyclic graph

Moralize a directed acyclic graph which means marrying parents and dropping directions.

Usage

moralize(object, ...)

## Default S3 method:
moralize(object, result = NULL, ...)

Examples

## A graphNEL object
g1 <- ug(~a:b + b:c + c:d + d:e + e:f + a:f + b:e)
x <- minimal_triang(g1)

## g2 is a triangulation of g1 but it is not minimal
g2 <- ug(~a:b:e:f + b:c:d:e)
x <- minimal_triang(g1, tobject=g2)

## An adjacency matrix
g1m <- ug(~a:b + b:c + c:d + d:e + e:f + a:f + b:e, result="matrix")
x <- minimal_triangMAT(g1m)
Arguments

- **object**: A directed acyclic graph represented either as a `graphNEL` object, an `igraph`, a (dense) matrix, a (sparse) `dgCMatrix`.
- **...**: Additional arguments, currently not used
- **result**: The representation of the moralized graph. When NULL the representation will be the same as the input object.

Value

A moralized graph represented either as a `graphNEL`, a dense matrix or a sparse `dgCMatrix`.

Note

The workhorse is the `moralizeMAT` function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

- `mcs`, `junction_tree`, `rip`, `ug`, `dag`

Examples

```r
daG <- dag(~me+ve, ~me+al, ~ve+al, ~al+an, ~al+st, ~an+st)
moralize(daG)

daG <- dag(~me+ve, ~me+al, ~ve+al, ~al+an, ~al+st, ~an+st, result="matrix")
moralizeMAT(daG)

if (require(igraph)){
  M <- matrix(c(1,2,3,3), nrow=2)
  G <- graph.edgelist(M)
  G
  V(G)$name
  moralize(G)
  }
```
Maximal prime subgraph decomposition

Description
Finding a junction tree representation of the MPD (maximal prime subgraph decomposition) of an undirected graph. The maximal prime subgraph decomposition of a graph is the smallest subgraphs into which the graph can be decomposed.

Usage
mpd(object, tobject = minimal_triang(object), details = 0)

## Default S3 method:
mpd(object, tobject = triangulate(object), details = 0)

mpdMAT(amat, tamat = minimal_triangMAT(amat), details = 0)

Arguments
object An undirected graph; a graphNEL object, an igraph or an adjacency matrix.
tobject Any minimal triangulation of object; a graphNEL object, an igraph or an adjacency matrix.
details The amount of details to be printed.
amat An undirected graph; a symmetric adjacency matrix
tamat Any minimal triangulation of object; a symmetric adjacency matrix

Value
A list with components "nodes", "cliques", "separators", "parents", "children", "nLevels". The component "cliques" defines the subgraphs.

Author(s)
Clive Bowsher <C.Bowsher@statslab.cam.ac.uk> with modifications by Søren Højsgaard, <sorenh@math.aau.dk>

References

See Also
mcs, mcsMAT, minimal_triang, minimal_triangMAT, rip, ripMAT, triangulate, triangulateMAT
Examples

```r
## Maximal prime subgraph decomposition - a graphNEL object
g1 <- ug(~ a:b + b:c + c:d + d:e + e:f + a:f + b:e)
if (interactive()) plot(g1)
x <- mpd(g1)

## Maximal prime subgraph decomposition - an adjacency matrix
g1m <- ug(~ a:b + b:c + c:d + d:e + e:f + a:f + b:e, result="matrix")
if (interactive()) plot(as(g1m, "graphNEL"))
x <- mpdMAT(g1m)
```

---

**graph-query**  
*Query a graph*

**Description**

Unified approach to query a graph about its properties (based partly on functionality from gRbase and functionality imported from RBGL).

**Usage**

```r
querygraph(object, op, set = NULL, set2 = NULL, set3 = NULL)
```

```r
qgraph(object, op, set = NULL, set2 = NULL, set3 = NULL)
```

```r
ancestors(set, object)
```

```r
ancestralSet(set, object)
```

```r
parents(set, object)
```

```r
children(set, object)
```

```r
closure(set, object)
```

```r
simplicialNodes(object)
```

```r
ancestralGraph(set, object)
```

```r
is.complete(object, set = NULL)
```

```r
is.decomposition(set, set2, set3, object)
```

```r
is.simplicial(set, object)
```
## Arguments

- **object**: A graph.
- **op**: The operation or query.
- **set, set2, set3**: Sets of nodes in graph.

## Description

Generate a random directed acyclic graph (DAG)

## Usage

```r
random_dag(V, maxpar = 3, wgt = 0.1)
```

## Arguments

- **V**: The set of vertices.
- **maxpar**: The maximum number of parents each node can have
- **wgt**: A parameter controlling how likely it is for a node to have a certain number of parents; see 'Details'.

## Details

If the maximum number of parents for a node is, say 3 and \( wgt = 0.1 \), then the probability of the node ending up with 0, 1, 2, 3 parents is proportional to \( 0.1^0, 0.1^1, 0.1^2, 0.1^3 \).

## Value

A graphNEL object.

## Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

## Examples

```r
dg <- random_dag(1:1000, maxpar=5, wgt=.9)
table(sapply(vpar(dg),length))

dg <- random_dag(1:1000, maxpar=5, wgt=.5)
table(sapply(vpar(dg),length))

dg <- random_dag(1:1000, maxpar=5, wgt=.1)
table(sapply(vpar(dg),length))
```
**graph-rip**

Create RIP ordering of the cliques of an undirected graph; create junction tree.

**Description**

A RIP (running intersection property) ordering of the cliques is also called a perfect ordering. If the graph is not chordal, then no such ordering exists.

**Usage**

```r
rip(object, ...)  
## Default S3 method:  
rip(object, root = NULL, nLevels = NULL, ...)  
ripMAT(amat, root = NULL, nLevels = rep(2, ncol(amat)))  
junction_tree(object, ...)  
## Default S3 method:  
junction_tree(object, nLevels = NULL, ...)  
junction_treeMAT(amat, nLevels = rep(2, ncol(amat)), ...)  
jTree(object, ...)  
```

**Arguments**

- `object`: An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
- `...`: Additional arguments; currently not used
- `root`: A vector of variables. The first variable in the perfect ordering will be the first variable on `root`. The ordering of the variables given in `root` will be followed as far as possible.
- `nLevels`: Typically, the number of levels of the variables (nodes) when these are discrete. Used in determining the triangulation using a "minimum clique weight heuristic". See section 'details'.
- `amat`: Adjacency matrix

**Details**

The RIP ordering of the cliques of a decomposable (i.e. chordal) graph is obtained by first ordering the variables linearly with maximum cardinality search (by `mcs`). The root argument is transferred to `mcs` as a way of controlling which clique will be the first in the RIP ordering. The `junction_tree()` (and `junction_tree()`) (for "junction tree") is just a wrapper for a call of `triangulate()` followed by a call of `rip()`.
Value

`rip` returns a list (an object of class `riponder`). A `print` method exists for such objects.

Synonymous functions

For backward compatibility with downstream packages we have the following synonymous functions:

- `jTree = junction_tree` (Used in `rags2ridges`)
- `junctionTree = junction_tree`

Note

The workhorse is the `ripMAT()` function. The `nLevels` argument to the `rip` functions has no meaning.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`mcs`, `triangulate`, `moralize`, `ug`, `dag`

Examples

```r
## graphNEL
mcs(uG)
rip(uG)
junction_tree(uG)

## Adjacency matrix
uG <- ug(~me:ve:al + al:an:st, result="matrix")
mcs(uG)
rip(uG)
junction_tree(uG)

## Sparse adjacency matrix
uG <- ug(c("me", "ve", "al"), c("al", "an", "st"), result="dgCMatrix")
mcs(uG)
rip(uG)
junction_tree(uG)

## Non-decomposable graph
uG <- ug(~1:2 + 2:3 + 3:4 + 4:5 + 5:1)
mcs(uG)
rip(uG)
junction_tree(uG)
```
Topological sort of vertices in directed acyclic graph

Description

A topological ordering of a directed graph is a linear ordering of its vertices such that, for every edge \((u \rightarrow v)\), \(u\) comes before \(v\) in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG). Any DAG has at least one topological ordering. Can hence be used for checking if a graph is a DAG.

Usage

topo_sort(object, index = FALSE)
topo_sortMAT(amat, index = FALSE)
topoSort(object, index = FALSE)
topoSortMAT(amat, index = FALSE)

Arguments

object An graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.
index If FALSE, an ordering is returned if it exists and character(0) otherwise. If TRUE, the index of the variables in an adjacency matrix is returned and -1 otherwise.
amat Adjacency matrix.

Value

If FALSE, an ordering is returned if it exists and character(0) otherwise. If TRUE, the index of the variables in an adjacency matrix is returned and -1 otherwise.

Synonymous functions

The functions topo_sort / topoSort are synonymous with topo_sortMAT / topoSortMAT. One of the groups may be deprecated in the future.

Note

The workhorse is the topo_sortMAT function which takes an adjacency matrix as input.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>
graph-triangulate

See Also
dag, ug

Examples

dagMAT <- dag(~a:b:c + c:d:e, result="matrix")
dagMATS <- as(dagMAT, "dgMatrix")
dagNEL <- as(dagMAT, "graphNEL")

topo_sort(dagMAT)
topo_sort(dagMATS)
topo_sort(dagNEL)

Description

This function will triangulate an undirected graph by adding fill-ins.

Usage

triangulate(object, ...)

## Default S3 method:
triangulate(object, nLevels = NULL, result = NULL, check = TRUE, ...)

triang_mcwh(object, ...)

triang_elo(object, ...)

triang(object, ...)

## Default S3 method:
triang(object, control = list(), ...)

## Default S3 method:
triang_mcwh(object, nLevels = NULL, result = NULL, check = TRUE, ...)

## Default S3 method:
triang_elo(object, order = NULL, result = NULL, check = TRUE, ...)

triangulateMAT amat, nLevels = rep(2, ncol(amat)), ...)

triang_MCWHMAT_(amat, nLevels = rep(2, ncol(amat)), ...)

triang_eloMAT_(amat, order)
triang_eloMAT(amat, order = NULL)

Arguments

object  An undirected graph represented either as a graphNEL object, an igraph, a (dense) matrix, a (sparse) dgCMatrix.

...  Additional arguments, currently not used.

nLevels  The number of levels of the variables (nodes) when these are discrete. Used in determining the triangulation using a "minimum clique weight heuristic". See section 'details'.

result  The type (representation) of the result. Possible values are "graphNEL", "igraph", "matrix", "dgCMatrix". Default is the same as the type of object.

check  If TRUE (the default) it is checked whether the graph is triangulated before doing the triangulation; gives a speed up if FALSE.

control  A list controlling the triangulation; see 'examples'.

order  Elimination order; a character vector or numeric vector.

amat  Adjacency matrix; a (dense) matrix, or a (sparse) dgCMatrix.

Details

There are two type of functions: triang and triangulate

The workhorse is the triangulateMAT function.

The triangulation is made so as the total state space is kept low by applying a minimum clique weight heuristic: When a fill-in is necessary, the algorithm will search for an edge to add such that the complete set to be formed will have as small a state-space as possible. It is in this connection that the nLevels values are used.

Default (when nLevels=NULL) is to take nLevels=2 for all nodes. If nLevels is the same for all nodes then the heuristic aims at keeping the clique sizes small.

Value

A triangulated graph represented either as a graphNEL, a (dense) matrix or a (sparse) dgCMatrix.

Note

Care should be taken when specifying nLevels for other representations than adjacency matrices: Since the triangulateMAT function is the workhorse, any other representation is transformed to an adjacency matrix and the order of values in nLevels must come in the order of the nodes in the adjacency matrix representation.

Currently there is no check for that the graph is undirected.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>
See Also

ug, dag, mcs, mcsMAT, rip, ripMAT, moralize, moralizeMAT

Examples

```
## graphNEL
uG1 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a)
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="matrix")
uG3 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="dgCMatrix")

## Default triangulation: minimum clique weight heuristic
# (default is that each node is given the same weight):

tuG1 <- triang(uG1)
## Same as
triang_mcwh(uG1)

## Alternative: Triangulation from a desired elimination order
# (default is that the order is order of the nodes in the graph):

triang(uG1, control=list(method="elo"))
## Same as:
triang_elo(uG1)

## More control: Define the number of levels for each node:

tuG1 <- triang(uG1, control=list(method="mcwh", nLevels=c(2, 3, 2, 6, 4, 9)))
tuG1 <- triang_mcwh(uG1, nLevels=c(2, 3, 2, 6, 4, 9))

tuG1 <- triang(uG1, control=list(method="elo", order=c("a", "e", "f")))
tuG1 <- triang_elo(uG1, order=c("a", "e", "f"))

## graphNEL
uG1 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a)
tuG1 <- triangulate(uG1)

## adjacency matrix
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="matrix")
tuG2 <- triangulate(uG2)

## adjacency matrix (sparse)
uG2 <- ug(~a:b + b:c + c:d + d:e + e:f + f:a, result="dgCMatrix")
tuG2 <- triangulate(uG2)
```

Description

List of vertices and their parents for graph.
Usage

vchi(object, getv = TRUE, forceCheck = TRUE)
vchiMAT(object, getv = TRUE, forceCheck = TRUE)
vpar(object, getv = TRUE, forceCheck = TRUE)
vparMAT(object, getv = TRUE, forceCheck = TRUE)

Arguments

object An object representing a graph. Valid objects are an adjacency matrix or as a
graphNEL.

getv The result is by default a list of vectors of the form (v,pa1,pa2,... paN) where
pa1,pa2,... paN are the parents of v. If getv is FALSE then the vectors will
have the form (pa1,pa2,... paN)

forceCheck Logical indicating if it should be checked that the object is a DAG.

Value

A list of vectors where each vector will have the form (v,pa1,pa2,... paN) where pa1,pa2,...
paN are the parents of v.

See Also

dag, ug

Examples

## DAGs
dagMAT <- dag(~a:b:c + c:d:e, result="matrix")
dagNEL <- dag(~a:b:c + c:d:e, result="graphNEL")
vpar(dagMAT)
vpar(dagNEL)
vpar(dagMAT, getv=FALSE)
vpar(dagNEL, getv=FALSE)

## Undirected graphs
ugMAT <- ug(~a:b:c + c:d:e, result="matrix")
ugNEL <- ug(~a:b:c + c:d:e, result="graphNEL")

## Not run:
## This will fail because the adjacency matrix is symmetric and the
## graphNEL has undirected edges
vpar(ugMAT)
vpar(ugNEL)

## End(Not run)
## When forceCheck is FALSE, it will not be detected that the graphs are undirected.
vpar(ugMAT, forceCheck=FALSE)
vpar(ugNEL, forceCheck=FALSE)
## Bidirected graphs

This is, for graphNELs, the same as working with bidirected edges:

```r
if (require(graph)){
  graph::edgemode(ugNEL)
  graph::edgemode(ugNEL) <- "directed"
  graph::edgemode(ugNEL)
  vpar(ugNEL,FALSE)
}
```

---

### Description

This package provides a basis for graphical modelling in R and in particular for other graphical modelling packages, most notably `gRim`, `gRain` and `gRc`.

### Details

**gRbase** provides the following:

- Implementation of various graph algorithms, including maximum cardinality search, maximal prime subgraph decomposition, triangulation. See the vignette `graphs`.
- Implementation of various "high level" array operations, including multiplication/division, marginalization, slicing, permutation. See the vignette `ArrayOps`.
- Implementation of various "low level" array operations. See the vignette `ArrayOpsPrim`.
- A collection of datasets
- A general framework for setting up data and model structures and provide examples for fitting hierarchical log linear models for contingency tables and graphical Gaussian models for the multivariate normal distribution. (Notice: This last part is not maintained / developed further.)

### Authors

Soren Hojsgaard, Department of Mathematical Sciences, Aalborg University, Denmark

Contributions from Claus Dethlefsen, Clive Bowsher, David Edwards.

### Acknowledgements

Thanks to the other members of the gR initiative, in particular to David Edwards for providing functions for formula-manipulation.

### References


grbase-utilities  gRbase utilities

Description

Various utility functions for gRbase. Includes 'faster versions' of certain standard R functions.

Usage

rhsFormula2list(form)
rhsf2list(form)
rhsf2vec(form)
listify_dots(dots)
list2rhsFormula(form)
list2rhsf(form)
rowmat2list(X)
colmat2list(X)
matrix2list(X, byrow = TRUE)
which.arr.index(X)
which_matrix_index(X)
rowSumsPrim(X)
colSumsPrim(X)
colwiseProd(v, X)
lapplyV2I(setlist, item)
lapplyI2V(setlist, item)

Arguments

form  Formula specification (a right-hand sided formula, a numeric/character vector or a list of vectors).
dots  dot-arguments to be turned into a list
X \quad \text{A matrix.}

byrow \quad \text{Should the split be by row or by column.}

v \quad \text{A vector.}

setlist \quad \text{A list of atomic vectors}

item \quad \text{An atomic vector}

Details

which.arr.ind: Returns matrix n x 2 matrix with indices of non-zero entries in matrix X. Notice which_matrix_index__ is cpp implementation.

colwiseProd: multiplies a vector v and a matrix X columnwise (as opposed to rowwise which is achieved by v * X). Hence colwiseProd does the same as t(v * t(X)) - but it does so faster for numeric values.

- lapplyV2I: same as but much faster than lapply(setlist, function(elt) match(elt, item))
- lapplyI2V: same as but faster than lapply(setlist, function(x) item[x])

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
## colwiseProd
X <- matrix(1:16, nrow=4)
v <- 1:4
t(v * t(X))
colwiseProd(v, X)
```

```r
## Not run:
system.time(for (ii in 1:100000) t(v * t(X)))
system.time(for (ii in 1:100000) colwiseProd(v, X))

## End(Not run)

setlist <- list(c(1,2,3), c(2,3,4), c(2,4,5))
item <- c(2,3)

lapplyV2I(setlist, item)
lapply(setlist, function(gg) match(gg, item))
lapplyI2V(setlist, item)
lapply(setlist, function(x) item[x])

if (require(microbenchmark)){
microbenchmark(
lapplyV2I(setlist, item),
lapply(setlist, function(elt) match(elt, item)))
microbenchmark::microbenchmark(
```
lapply12v(setlist, item),
lapply(setlist, function(elt) item[elt])
}

---

**grbase_generic**

*Compile and propagate functions*

**Description**

`compile` and `propagate` are generic functions which invoke particular methods which depend on the class of the first argument.

**Usage**

```r
fit(object, ...)
```

```r
compile(object, ...)
```

```r
propagate(object, ...)
```

```r
stepwise(object, ...)
```

**Arguments**

- `object` An object
- `...` Additional arguments which depends on the class of the object

**Value**

The value returned depends on the class of the first argument.

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**References**


---

**internal**

*Internal functions for the gRbase package*

**Description**

These functions are not intended to be called directly.
**set-operations**

**Suite of set operations**

---

**Description**

Set operations for gRbase and related packages.

**Usage**

- `maximal_sets(setlist, index = FALSE)`
- `minimal_sets(setlist, index = FALSE)`
- `remove_redundant(setlist, maximal = TRUE, index = FALSE)`
- `is_inset(x, setlist, index = FALSE)`
- `get_subset(x, setlist, all = FALSE)`
- `get_superset(x, setlist, all = FALSE)`
- `is_subsetof(set, set2)`
- `subsetof(x, set)`

**Arguments**

- `setlist` List of vectors (representing a set of subsets)
- `index` Logical; should indices (in setlist) be returned or a set of subsets.
- `maximal` Logical; see section 'Details' for a description.
- `x, set, set2` Vector representing a set.
- `all` Logical; see section 'Details' for a description.

**Details**

- 'setlist' is a list of vectors representing a set of subsets; i.e. \( V_1, \ldots, V_Q \) where \( V_k \) is a subset of some base set \( V \).
- 'all' If true, `get_superset` will return index of all vectors containing the element; otherwise only the first index is returned.
- `is_inset`: Checks if the set \( x \) is in one of the \( V_k \)'s.
- `remove_redundant`: Returns those \( V_k \) which are not contained in other subsets; i.e. gives the maximal sets. If maximal is FALSE then returns the minimal sets; i.e. \( V_k \) is returned if \( V_k \) is contained in one of the other sets \( V_l \) and there are no set \( V_n \) contained in \( V_k \).
Notice that the comparisons are made by turning the elements into characters and then comparing these. Hence 1 is identical to "1".

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**Examples**

```r
set <- list(c(1, 2), c(1, 2, 3), c(2, 3, 6), c(2, 4), c(5, 6), 5)

e11 <- c(2, 1)
e12 <- c(2, 3)
e13 <- c(4, 3)
e14 <- c(2, 1, 3)

maximal_sets(set)
minimal_sets(set)

remove_redundant(set)
remove_redundant(set, maximal=FALSE)

is_inset(e11, set)
is_inset(e12, set)
is_inset(e13, set)

get_subset(e11, set)
get_subset(e11, set)
get_subset(e12, set)
get_subset(e13, set)

get_superset(e11, set)
get_superset(e11, set, all=TRUE)
get_superset(e12, set)
get_superset(e13, set)

is_subsetof(e11, e11)
is_subsetof(e11, e12)
is_subsetof(e11, e14)
```

---

**ug2dag**

*Coerce between undirected and directed graphs when possible*

**Description**

An undirected graph $G$ can be converted to a dag if $G$ is chordal. A dag $D$ can be converted to an undirected graph if $D$ can be moralized without adding edges.
ug2dag

Usage

ug2dag(gn)

Arguments

gn  A graphNEL object or an object that can be converted to a graphNEL object.
Index

* datasets
  data-ashtrees, 22
  data-BodyFat, 23
  data-breastcancer, 24
  data-cad, 25
  data-carass, 26
  data-chestSim, 27
  data-dietox, 28
  data-dumping, 29
  data-lizard, 29
  data-mathmark, 30
  data-mildew, 31
  data-milkcomp, 32
  data-Nutrimouse, 33
  data-personality, 34
  data-rats, 35
  data-reinis, 35
  data-wine, 36

* graphics
  graph-iplot, 49

* graphs
  gRbase, 67

* models
  graph-gcproperties, 47
    gRbase, 67

* multivariate
  gRbase, 67

* utilities
  api-parray, 9
  api-tabDist, 12
  api-tabNew, 14
  array-simulate, 18
  compareModels, 20
  cov2pcor, 20
  fastcombn, 37
  graph-clique, 39
  graph-coerce-api, 41
  graph-create, 44
  graph-gcproperties, 47
  graph-is, 49
  graph-mcs, 52
  graph-min-triangulate, 54
  graph-moralize, 55
  graph-mpd, 57
  graph-randomdag, 59
  graph-rip, 60
  graph-toposort, 62
  graph-triangulate, 63
  gRbase.generics, 70
  %>% (internal), 70
  %a*% (api-pct-operations), 11
  %a+% (api-pct-operations), 11
  %a-% (api-pct-operations), 11
  %a/0% (api-pct-operations), 11
  %a/% (api-pct-operations), 11
  %a==% (api-pct-operations), 11
  %a_% (api-pct-operations), 11
  %a^% (api-pct-operations), 11
  %aalign% (api-pct-operations), 11
  %amarg% (api-pct-operations), 11
  %aperm% (api-pct-operations), 11
  %aslice*% (api-pct-operations), 11
  %aslice% (api-pct-operations), 11

  addEdge.gModel (internal), 70
  all_pairs, 3
  all_subsets, 4
  all_subsets0 (all_subsets), 4
  ancestors (graph-query), 58
  ancestralGraph (graph-query), 58
  ancestralSet (graph-query), 58
  api-array-07, 4
  api-array-properties, 6
  api-cell, 7
  api-cell., 8
  api-parray, 9
  api-pct-operations, 11
  api-tabDist, 12
  api-tabNew, 14
INDEX

api-tabX, 15
api-tabX, 16
api_tabSlice, 17
array-simulate, 18
as.adjMAT (gmwr_book), 38
as.parray (api-parray), 9
ashtrees (data-ashtrees), 22
BodyFat (data-BodyFat), 23
breastcancer (data-breastcancer), 24
cad1 (data-cad), 25
cad2 (data-cad), 25
carcass (data-carcass), 26
carcassall (data-carcass), 26
cell2entry (api-cell), 7
cell2entry_(api-cell_), 8
cell2entry_perm (api-cell), 7
cell2entry_perm_(api-cell_), 8
chestSim100 (data-chestSim), 27
chestSim1000 (data-chestSim), 27
chestSim10000 (data-chestSim), 27
chestSim5000 (data-chestSim), 27
chestSim50000 (data-chestSim), 27
children (graph-query), 58
closure (graph-query), 58
cooereGraph (graph-coerce), 40
colmat2list (grbase-utilities), 68
colSumsPrim (grbase-utilities), 68
colwiseProd (grbase-utilities), 68
combn, 38
combn_prim (fastcombn), 37
compareModels, 20
compile (grbase_generics), 70
c onc2pcor (cov2pcor), 20
cov2pcor, 20
dag, 40, 43, 51, 53, 56, 61, 63, 65, 66
dag (graph-create), 44
dag2chol, 21
dagi (graph-create), 44
dagList (graph-create), 44
data-ashtrees, 22
data-BodyFat, 23
data-breastcancer, 24
data-cad, 25
data-carcass, 26
data-chestSim, 27
data-dietox, 28
data-dumping, 29
data-lizard, 29
data-mathmark, 30
data-mildew, 31
data-milkcomp, 32
data-Nutrimouse, 33
data-personality, 34
data-rats, 35
data-reinis, 35
data-wine, 36
data2parray (api-parray), 9
dietox (data-dietox), 28
dimnames_match (api-array-properties), 6
downstream-aliases, 37
dropEdge.gModel (internal), 70
dumping (data-dumping), 29
dgeList (graph-edgeList), 46
dgeListMAT (graph-edgeList), 46
dell (downstream-aliases), 37
dellK (downstream-aliases), 37
demand (api-cell), 7
demand_(api-cell_), 8
extract.power (internal), 70
fact_grid (api-cell), 7
fastcombn, 37
fit (grbase_generics), 70
g_adl2dm_ (graph-coerce-list), 43
g_adl2gn_ (graph-coerce-list), 43
g_adl2ig_ (graph-coerce-list), 43
g_adl2M_ (graph-coerce-list), 43
g_adl2sm_ (graph-coerce-list), 43
g_adl2XX_ (graph-coerce-list), 43
g_dagl2dm_ (graph-coerce-list), 43
g_dagl2gn_ (graph-coerce-list), 43
g_dagl2ig_ (graph-coerce-list), 43
g_dagl2M_ (graph-coerce-list), 43
g_dagl2sm_ (graph-coerce-list), 43
g_dagl2XX_ (graph-coerce-list), 43
g_dm2gn_ (graph-coerce-api), 41
g_dm2ig_ (graph-coerce-api), 41
g_dm2sm_ (graph-coerce-api), 41
g_gn2dm_ (graph-coerce-api), 41
g_gn2ftM_ (graph-coerce-api), 41
g_gn2ig_ (graph-coerce-api), 41
g_gn2sm_ (graph-coerce-api), 41
g_gn2tfM_ (graph-coerce-api), 41
INDEX

g_M2dagl_ (graph-coerce-api), 41

g_ig2dm_ (graph-coerce-api), 41

g_ig2gn_ (graph-coerce-api), 41

g_ig2sm_ (graph-coerce-api), 41

get_cliques (graph-clique), 39

get_superset (set-operations), 71

glm (internal), 70

gmwr_book, 38

graph-coerce-api, 41

graph-coerce-api, 41

graph-coerce-api, 41

graph-coerce-api, 41

graph-mpd, 57

graph-query, 58

graph-randomdag, 59

graph-rip, 60

graph-toposort, 62

graph-triangulate, 63

graph-vpar, 65

graph_as (graph-coerce), 40

graphNEL2M (graph-coerce-api), 41

grBase, 67

grbase-utilities, 68

gbaseGenerics, 70

internal, 70

intersectPrim (internal), 70

iplot (graph-plot), 49

is.adjMAT (graph-is), 49

is.complete (graph-query), 58

is.DAG (graph-is), 49

is.decomposition (graph-query), 58

is.DG (graph-is), 49

is.named.array, 10

is.named.array (api-array-properties), 6

is.simplicial (graph-query), 58

is.subsetof (set-operations), 71

is.TUG (graph-is), 49

is.UG (graph-is), 49

is_adjMAT (graph-is), 49

is_dag (graph-is), 49

is_dagMAT (graph-is), 49

is_dg (graph-is), 49

is_dgMAT (graph-is), 49

is_dimnames_ (api-array-properties), 6

is_inset (set-operations), 71

is_named_array_ (api-array-properties), 6

is_number_vector_ (api-array-properties), 6

is_subsetof (set-operations), 71

is_tug (graph-is), 49

is_tugMAT (graph-is), 49

is_uG (graph-is), 49

is_ugMAT (graph-is), 49

isDecomposable (graph-gcproperties), 47

isGraphical (graph-gcproperties), 47

isGSD_glist (internal), 70

jTree (graph-rip), 60

junction_tree, 53, 56

junction_tree (graph-rip), 60

junction_treeMAT (graph-rip), 60

junctionTree (graph-rip), 60

lapply12V (grbase-utilities), 68

lapply2V (grbase-utilities), 68

list2rhs (grbase-utilities), 68

list2rhsFormula (grbase-utilities), 68
INDEX

listify dots (grbase-utilities), 68
lizard (data-lizard), 29
lizardAGG (data-lizard), 29
lizardRAW (data-lizard), 29

M2graphNEL (graph-coerce-api), 41
make levels (api-cell), 8
makeDimNames (api-cell), 9
MAT2ftM (internal), 70
matchPrim (internal), 70
math (data-mathmark), 30
mathmark (data-mathmark), 30
matrix2list (grbase-utilities), 68
max_cliqueMAT (graph-clique), 39
maxCliqueMAT (graph-clique), 39
maximal sets (set-operations), 71
mcs, 40, 48, 56, 57, 61, 65
mcs (graph-mcs), 52
mcs_marked (graph-mcs), 52
mcs_markedMAT (graph-mcs), 52
mcsMAT, 40, 57, 65
mcsMAT (graph-mcs), 52
mildew (data-mildew), 31
milkcomp (data-milkcomp), 32
milkcomp1 (data-milkcomp), 32
minimal sets (set-operations), 71
minimal_triang, 57
minimal_triang (graph-min-triangulate), 54
minimal_triangMAT, 57
minimal_triangMAT (graph-min-triangulate), 54
moralize, 40, 53, 61, 65
moralize (graph-moralize), 55
moralizeMAT, 40, 65
moralizeMAT (graph-moralize), 55
mpd, 55
mpd (graph-mpd), 57
mpdMAT (graph-mpd), 57

names2pairs (all_pairs), 3
next cell (api-cell), 7
next cell2 (api-cell), 7
next cell2 (api-cell), 7
next cell (api-cell), 7
next cell_slice (api-cell), 7
next cell_slice (api-cell), 7
nonEdgeList (graph-edgeList), 46
nonEdgeListMAT (graph-edgeList), 46
Nutrimouse (data-Nutrimouse), 33
outerPrim (internal), 70
pairs2num (grbase-utilities), 68
parents (graph-query), 58
parray (api-parray), 9
perm_cell_entries (api-cell), 7
perm_cell_entries (api-cell), 7
personality (data-personality), 34
processFormula (internal), 70
propagate (grbase_generics), 70
qgraph (graph-query), 58
querygraph (graph-query), 58
random_dag (graph-randomdag), 59
randomGraph (internal), 70
rats (data-rats), 35
reinis (data-reinis), 35
remove_redundant (set-operations), 71
rhsf2list (grbase-utilities), 68
rhsf2vec (grbase-utilities), 68
rhsFormula2list (grbase-utilities), 68
rip, 40, 48, 53, 55–57, 65
rip (graph-rip), 60
ripMAT, 40, 57, 65
ripMAT (graph-rip), 60
rowmat2list (grbase-utilities), 68
rowSumsPrim (grbase-utilities), 68
selectOrder (internal), 70
set-operations, 71
setdiffPrim (internal), 70
simplicialNodes (graph-query), 58
simulate.array (array-simulate), 18
simulate.table (array-simulate), 18
simulate.xtabs (array-simulate), 18
simulateArray (array-simulate), 18
slice2entry (api-cell), 7
slice2entry (api-cell), 7
solveSPD (internal), 70
stepwise (grbase_generics), 70
subsetof (set-operations), 71
symMAT2ftM (internal), 70

tab_add (api-tabX), 16
tab_align (api-tabX), 16
tab_div0 (api-tabX), 16
tab_div (api-tabX), 16
INDEX

tab_equal_(api-tabX_), 16
tab_expand_(api-tabX_), 16
tab_list_add_(api-tabX_), 16
tab_list_mult_(api-tabX_), 16
tab_marg_(api-tabX_), 16
tab_mult_(api-tabX_), 16
tab_op_(api-tabX_), 16
tab_perm_(api-tabX_), 16
tab_subt_(api-tabX_), 16
tabAdd_(api-tabX), 15
tabAlign_(api-tabX), 15
tabDist_(api-tabDist), 12
tabDiv_(api-tabX), 15
tabDiv0_(api-tabX), 15
tabEqual_(api-tabX), 15
tabExpand_(api-tabX), 15
tableDiv_(api-array-07), 4
tableGetSliceIndex_(api-array-07), 4
tableMargin_(api-array-07), 4
tableMult_(api-array-07), 4
tableOp_(api-array-07), 4
tableOp0_(api-array-07), 4
tableOp2_(api-array-07), 4
tablePerm_(api-array-07), 4
tableSetSliceValue_(api-array-07), 4
tableSlice_(api-array-07), 4
tableSlicePrim_(api-array-07), 4
tabListAdd_(api-tabX), 15
tabListMult_(api-tabX), 15
tabMarg_(api-tabX), 15
tabMult_(api-tabX), 15
tabNew_(api-tabNew), 14
tabNormalize_(api-tabX), 15
tabOp_(api-tabX), 15
tabPerm_(api-tabX), 15
tabProd_(api-tabX), 15
tabSlice_(api_tabSlice), 17
tabSlice2_(api_tabSlice), 17
tabSlice2Entries_(api_tabSlice), 17
tabSliceMult_(api_tabSlice), 17
tabSlicePrim_(api_tabSlice), 17
tabSubt_(api-tabX), 15
tabSum_(api-tabX), 15
topo_sort_(graph-toposort), 62
topo_sortMAT_(graph-toposort), 62
topoSort_(graph-toposort), 62
topoSortMAT_(graph-toposort), 62
triang_(graph-triangulate), 63

ug, 40, 43, 51, 53, 56, 61, 63, 65, 66
ug_(graph-create), 44
ug2dag, 72
ug1_(graph-create), 44
ugList_(graph-create), 44
uniquePrim_(internal), 70
unlistPrim_(internal), 70

vchi_(graph-vpar), 65
vchiMAT_(graph-vpar), 65
vpar_(graph-vpar), 65
vparMAT_(graph-vpar), 65

which.arr.index_(grbase-utilities), 68
which_matrix_index_(grbase-utilities), 68

wine_(data-wine), 36