Package ‘gRbase’

July 3, 2020

Version 1.8-6.7

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
Dethlefsen and Højsgaard, (2005, <doi:10.18637/jss.v014.i17>).
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 http://people.math.aau.dk/~sorenh/software/gR/

ByteCompile Yes

Encoding UTF-8

VignetteBuilder knitr

Depends R (>= 3.6.0), methods

Imports graph, Rgraphviz, RBGL, stats4, igraph, magrittr, Matrix, Rcpp
(>= 0.11.1)

Suggests testthat (>= 2.1.0), microbenchmark, knitr

biocViews

LinkingTo Rcpp (>= 0.11.1), RcppEigen, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-07-03 16:50:08 UTC

R topics documented:

`all_pairs` *Create all possible pairs*

Description

Create all possible pairs of two character vectors.

Usage

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

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 wil also be obtained.

Author(s)

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

Examples

```
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

```
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.

api-array-properties *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

```
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

```
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
```

```
ar2
## 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")))
ar2
## 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)
```

`api-cell` *Table cell operations.*

Description

Low level table cell operations.

Usage

```
cell2entry(cell, dim)

entry2cell(entry, dim)

next_cell(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))
```

---

api-cell_ *Low level table cell operations implemented in c++*

Description

Corresponding R functions without the trailing underscore exist.

Usage

```
cell2entry_(cell, dim)

make_plevels_(dim)

entry2cell_(entry, dim)

next_cell_(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

`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).

`api-parray` *Representation of and operations on multidimensional arrays*

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

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

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 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 `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 parrays 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(UCBAdmissions,c(1,2), normalize="first")
```

`api-pct-operations` *Array algebra*

Description

Addition, subtraction etc. of arrays

Usage

```
a1 %a+% a2

a1 %a-% a2

a1 %a*% a2

a1 %a/% a2

a1 %a/0% a2

tab1 %ap% perm

tab1 %a_% marg

tab1 %a==% tab2

tab1 %a^% extra

tab1 %aa% tab2
```

Arguments

`tab1, tab2`	Multidimensional 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 permutiation.
`marg`	A vector of indices or dimnames or a right hand sided formula giving the desired marginal.
`extra`	List defining the extra dimensions.
`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.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

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)

tabDist(hec, marg= ~Hair + Eye, normalize=FALSE)

## 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)
```

---

api-tabNew *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 `smooth` is positive then `smooth` is added to `values` before normalization takes place.

Value

An array.

Author(s)

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

Examples

```
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
```

api-tabX *Interface - operations on multidimensional arrays.*

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.

---

api-tabX_ *Table operations implemented in c++*

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)

tab_list_mult_(lst)

tab_list_add_(lst)
```

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.

`api_tabSlice` *Array slices*

Description

Functions for extracting slices of arrays

Usage

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

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

tabSlicePrim(tab, slice, drop = TRUE)

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

tabSlice2Entries(tab, slice, complement = FALSE)
```

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.

Author(s)

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

Examples

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

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

tabSlice2Entries(x, slice=s)
tabSlice2Entries(x, slice=s, complement=TRUE)

## ar_slice_mult
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

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))
```

`compareModels` *Generic function for model comparison*

Description

`compareModels` is a generic functions which invoke particular methods which depend on the class of the first argument

Usage

```
compareModels(object, object2, ...)
```

Arguments

`object, object2`	Model objects
`...`	Additional arguments

Value

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

Author(s)

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

`cov2pcor` *Partial correlation (matrix)*

Description

`cov2pcor` calculates the partial correlation matrix from an (empirical) covariance matrix while `conc2pcor` calculates the partial correlation matrix from a concentration matrix (inverse covariance matrix).

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)
```

`data-ashtrees` *Crown dieback in ash trees*

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

Skovgaard JP, Thomsen IM, Skovgaard IM and Martinussen T (2009). Associations among symptoms of dieback in even-aged stands of ash (Fraxinus excelsior L.). Forest Pathology.

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 http://lib.stat.cmu.edu/datasets/bodyfat

References

Bailey, Covert (1994). _Smart Exercise: Burning Fat, Getting Fit_, Houghton-Mifflin Co., Boston, pp. 179-186.

Behnke, A.R. and Wilmore, J.H. (1974). _Evaluation and Regulation of Body Build and Composition_, Prentice-Hall, Englewood Cliffs, N.J.

Siri, W.E. (1956), "Gross composition of the body", in _Advances in Biological and Medical Physics_, vol. IV, edited by J.H. Lawrence and C.A. Tobias, Academic Press, Inc., New York.

Katch, Frank and McArdle, William (1977). _Nutrition, Weight Control, and Exercise_, Houghton Mifflin Co., Boston.

Wilmore, Jack (1976). _Athletic Training and Physical Fitness: Physiological Principles of the Conditioning Process_, Allyn and Bacon, Inc., Boston.

Examples

```
data(BodyFat)
head(BodyFat)
```

data-breastcancer *Gene expression signatures for p53 mutation status in 250 breast cancer samples*

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

Miller et al (2005, PubMed ID:16141321)

Examples

```
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.

Usage

```
data(cad1)
```

Format

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

`Sex` a factor with levels `Female` `Male`

`AngPec` a factor with levels `Atypical` `None` `Typical`

`AMI` a factor with levels `Definite` `NotCertain`

`QWave` a factor with levels `No` `Yes`

`QWavecode` a factor with levels `Nonusable` `Usable`

`STcode` a factor with levels `Nonusable` `Usable`

`STchange` a factor with levels `No` `Yes`

`SuffHeartF` a factor with levels `No` `Yes`

`Hypertrophi` a factor with levels `No` `Yes`

`Hyperchol` a factor with levels `No` `Yes`

`Smoker` a factor with levels `No` `Yes`

`Inherit` a factor with levels `No` `Yes`

`Heartfail` a factor with levels `No` `Yes`

`CAD` a factor with levels `No` `Yes`

Details

* cad1: Complete dataset, 236 cases.

* cad2: Incomplete dataset, 67 cases. Information on (some of) the variables Hyperchol, Smoker, Inherit is missing.

References

Højsgaard, Søren and Thiesson, Bo (1995). BIFROST - Block recursive models Induced From Relevant knowledge, Observations and Statistical Techniques. Computational Statistics and Data Analysis, vol. 19, p. 155-175

Hansen, J. F. (1980). The clinical diagnoisis of ichaeme heart disease du to coronary artery disease. Danish Medical Bulletin

Examples

```
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 auxillary 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

`slhouse` 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

Busk, H., Olsen, E. V., Brøndum, J. (1999) Determination of lean meat in pig carcasses with the Autofom classification system, Meat Science, 52, 307-314

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.

Usage

```
data(chestSim500)
```

Format

A data frame with 500 observations on the following 8 variables.

`asia` a factor with levels `yes` `no`

`tub` a factor with levels `yes` `no`

`smoke` a factor with levels `yes` `no`

`lung` a factor with levels `yes` `no`

`bronc` a factor with levels `yes` `no`

`either` a factor with levels `yes` `no`

`xray` a factor with levels `yes` `no`

`dysp` a factor with levels `yes` `no`

References

Lauritzen and Spiegelhalter (1988) Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems (with Discussion). J. Roy. Stat. Soc. 50, p. 157-224.

Examples

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

data-dietox *Growth curves of pigs in a 3x3 factorial experiment*

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

Lauridsen, C., Højsgaard, S., Sørensen, M.T. C. (1999) Influence of Dietary Rapeseed Oli, Vitamin E, and Copper on Performance and Antioxidant and Oxidative Status of Pigs. J. Anim. Sci.77:906-916

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% of gastric tissue), vagotomy and hemigastrectomy (removal of 50% of gastric tissue), and gastric restriction (removal of 75% of gastric tissue).

Source

Grizzle JE, Starmer CF, Koch GG (1969) Analysis of categorical data by linear models. Biometrics 25(3):489-504.

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). 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

Schoener TW (1968) The anolis lizards of bimini: Resource partitioning in a complex fauna. Ecology 49:704-726

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

David Edwards, An Introduction to Graphical Modelling, Second Edition, Springer Verlag, 2000

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

The format is: table [1:2, 1:2, 1:2, 1:2, 1:2, 1:2] 0 0 0 0 3 0 1 0 0 1 ... - attr(*, "dimnames")=List of 6 ..$ la10: chr [1:2] "1" "2" ..$ locc: chr [1:2] "1" "2" ..$ mp58: chr [1:2] "1" "2" ..$ c365: chr [1:2] "1" "2" ..$ p53a: chr [1:2] "1" "2" ..$ a367: chr [1:2] "1" "2"

References

Christiansen, S.K., Giese, H (1991) Genetic analysis of obligate barley powdery mildew fungus based on RFLP and virulence loci. Theor. Appl. Genet. 79:705-712

Examples

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

`data-milkcomp` *Milk composition data*

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

Charlotte Lauridsen and Viggo Danielsen (2004): Lactational dietary fat levels and sources influence milk composition and performance of sows and their progeny Livestock Production Science 91 (2004) 95-105

Examples

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

`data-Nutrimouse` *The Nutrimouse Dataset*

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, French.

References

Martin, P. G. P., Guillou, H., Lasserre, F., D<e9>jean, S., Lan, A., Pascussi, J.-M., San Cristobal, M., Legrand, P., Besse, P. and Pineau, T. (2007). Novel aspects of PPARa-mediated regulation of lipid and xenobiotic metabolism revealed through a multrigenomic study. Hepatology 54, 767-777.

Examples

```
data(Nutrimouse)
```

`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

Morrison, D.F. (1976). Multivariate Statistical Methods. McGraw-Hill, USA.

Edwards, D. (1995). Introduction to Graphical Modelling, Springer-Verlag. New York.

`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

A table with 6 discrete variables. A: smoking, B: strenous mental work, D: strenuous physical work, E: systolic blood pressure, F: ratio of lipoproteins, G: Family anamnesis of coronary heart disease.

References

Edwards and Havranek (1985): A fast procedure for model search in multidimensional contingency tables. Biometrika, 72: 339-351.

Reinis et al (1981): Prognostic significance of the risk profile in the prevention of coronary heart disease. Bratis. lek. Listy. 76: 137-150.

`data-wine` *Chemical composition of wine*

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
`Clri`	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

Frank, A. & Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

References

See references at `http://archive.ics.uci.edu/ml/datasets/Wine`

Examples

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

`downstream-aliases` *Downstream aliases*

Description

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

Usage

```
ar_prod_list(lst)
```

Arguments

`lst`	A list of arrays

`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

```
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
  )
}
```

`gmwr_book` *Functions from Graphical Modelling with R book*

Description

Functions that must be retained to make code from gmwr-book work

Usage

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

Arguments

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

`graph-clique` *Get cliques of an undirected graph*

Description

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

Usage

```
get_cliques(object)

max_cliqueMAT(amat)

getCliques(object)

maxCliqueMAT(amat)
```

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 triangualted 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

```
## 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)
```

graph-coerce *Graph coercion*

Description

Methods for changing graph representations

Usage

```
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

coerceGraph 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
## getCliques(g1)         ## Works

l1 <- list(c("a" ,"b"), c("b", "c"))
graph_as(l1, "graphNEL", "ugList")
```

---

graph-coerce-api *API for coercing graph representations*

Description

API for coercing graph representations.

Usage

```
g_gn2dm_(object)

g_gn2sm_(object)

g_gn2ig_(object)

g_dm2gn_(object)

g_dm2sm_(object)

g_dm2ig_(object)

g_sm2gn_(object)

g_sm2dm_(object)

g_sm2ig_(object)

g_ig2gn_(object)
```

```
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

graph-coerce-list *Coercion of graphs represented as lists*

Description

Coercion of graphs represented as lists to various graph formats.

Usage

```
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 (v1,...,vn) means that there will be arrows from v2,...,vn to v1.
`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

```
## 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
```

`graph-create` *Create undirected and directed graphs*

Description

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

Usage

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

ugList(x, result = "graphNEL")

dag(..., result = "graphNEL", 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 acyclig 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)
edgeList(gm)
edgeListMAT(gm)
edgeList(g, matrix=TRUE)
edgeList(gm, matrix=TRUE)
edgeListMAT(gm, matrix=TRUE)
```

```
nonEdgeList(g)
nonEdgeList(gm)
nonEdgeListMAT(gm)
## A graph without edges
g  <- ug(~a + b + c)
gm <- as(g, "matrix")
edgeList(g)
edgeList(gm)
edgeListMAT(gm)
edgeList(g, matrix=TRUE)
edgeList(gm, matrix=TRUE)
edgeListMAT(gm, matrix=TRUE)
nonEdgeList(g)
nonEdgeList(gm)
nonEdgeListMAT(gm)
```

graph-gcproperties *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 `g1 = ~a:b + a:c + b:c + c:d` has edges `ab,ac,bc,cd`. The cliques of this graph are `abc,cd`. Hence there is not a 1-1-correspondance between the graph and the generators.

On the other hand, `g2 <-~a:b:c + c:d` induces the same graph in this case there is a 1-1-correspondance.

The graph induced by `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

mcs, rip

Examples

```
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 )
```

`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

```
iplot(x, ...)

## S3 method for class 'graphNEL'
iplot(x, ...)
```

Arguments

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

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```
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 seach.

* 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 bidrected 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

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

## Undirected graphs
ugNEL  <- ug(~a:b:c + 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 `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 `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 `mcsMAT` function.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

See Also

`moralize`, `junction_tree`, `rip`, `ug`, `dag`

Examples

```
uG <- ug(~ me:ve + me:al + ve:al + al:an + al:st + an:st)
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 >= 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

```
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 superflous 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 the graph with the smallest number of fill-ins. The minimum triangulation is unique. Finding the minimum triangulation is NP-hard.

Value

`minimal_triang()` returns a graphNEL object while `minimal_triangMAT()` returns an adjacency matrix.

Author(s)

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

References

Kristian G. Olesen and Anders L. Madsen (2002): Maximal Prime Subgraph Decomposition of Bayesian Networks. IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B: CYBERNETICS, VOL. 32, NO. 1, FEBRUARY 2002

See Also

`mpd`, `rip`, `triangulate`

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)
```

`graph-moralize` *Moralize a directed acyclic graph*

Description

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

Usage

```
moralize(object, ...)

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

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

```
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)
}
```

---

graph-mpd *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

Kristian G. Olesen and Anders L. Madsen (2002): Maximal Prime Subgraph Decomposition of Bayesian Networks. IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PART B: CYBERNETICS, VOL. 32, NO. 1, FEBRUARY 2002

See Also

mcs, mcsMAT, minimal_triang, minimal_triangMAT, rip, ripMAT, triangulate, triangulateMAT

Examples

```
## 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

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

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

ancestors(set, object)

ancestralSet(set, object)

parents(set, object)

children(set, object)

closure(set, object)

simplicialNodes(object)
```

```
ancestralGraph(set, object)

is.complete(object, set = NULL)

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

is.simplicial(set, object)
```

Arguments

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

`graph-randomdag` *Random directed acyclic graph*

Description

Generate a random directed acyclic graph (DAG)

Usage

```
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

```
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

```
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 transfered 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 `ripOrder`. 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

```
## graphNEL
uG <- ug(~me:ve + me:al + ve:al + al:an + al:st + an:st)
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)
```

graph-toposort *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->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>

See Also

dag, ug

Examples

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

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

`graph-triangulate` *Triangulation of an undirected graph*

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`	Elimation 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` most 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)
```

graph-vpar *List of vertices and their parents for graph.*

Description

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

gRbase *The package 'gRbase': summary information*

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

Hojsgaard, S., Edwards, D., Lauritzen, S. (2012) Graphical models with R. Springer. ISBN: 978-1-4614-2298-3

Lauritzen, S. L. (2002). gRaphical Models in R. *R News*, 3(2)39.

`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`	A matrix.
`byrow`	Should the split be by row or by column.
`v`	A vector.
`setlist`	A list of atomic vectors
`item`	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(elt) item[elt])‘

Author(s)

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

Examples

```
## colwiseProd
X <- matrix(1:16, nrow=4)
v <- 1:4
t(v * t(X))
colwiseProd(v, X)
## 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(
  lapplyI2V(setlist, item),
  lapply(setlist, function(elt) item[elt]))
}
```

---

grbase_generics *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

```
fit(object, ...)

compile(object, ...)

propagate(object, ...)

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

Højsgaard, Søren; Edwards, David; Lauritzen, Steffen (2012): Graphical Models with R, Springer

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)

is.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. V1,...VQ where Vk 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 Vk's.

`remove_redundant`: Returns those Vk which are not contained in other subsets; i.e. gives the maximal sets. If maximal is FALSE then returns the minimal sets; i.e. Vk is returned if Vk is contained in one of the other sets Vl and there are no set Vn contained in Vk.

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

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

el1 <- c(2, 1)
el2 <- c(2, 3)
el3 <- c(4, 3)
el4 <- c(2, 1, 3)

maximal_sets(set)
minimal_sets(set)

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

is_inset(el1, set)
is_inset(el2, set)
is_inset(el3, set)

get_subset(el1, set)
get_subset(el1, set)
get_subset(el2, set)
get_subset(el3, set)

get_superset(el1, set)
get_superset(el1, set, all=TRUE)
get_superset(el2, set)
get_superset(el3, set)

is_subsetof(el1, el1)
is_subsetof(el1, el2)
is_subsetof(el1, el4)
```

`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.

Usage

```
ug2dag(gn)
```

Arguments

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

Index