Package ‘gRain’

October 10, 2024

Version 1.4.4

Title Graphical Independence Networks

Maintainer Søren Højsgaard <sorenh@math.aau.dk>

Description Probability propagation in Bayesian networks, also known as graphical independence networks. Documentation
of the package is provided in vignettes included in the package and in
the paper by Højsgaard (2012, <doi:10.18637/jss.v046.i10>).
See 'citation(``gRain'')' for details.

License GPL (>= 2)

Depends R (>= 4.2.0), methods, gRbase (>= 2.0.2),

Suggests bnlearn, gRim, knitr, markdown, microbenchmark, testthat (>= 2.1.0)

Imports igraph, stats4, broom, Rcpp (>= 0.11.1)

URL https://people.math.aau.dk/~sorenh/software/gR/

Encoding UTF-8

VignetteBuilder knitr

LazyLoad true

LinkingTo Rcpp (>= 0.11.1), RcppArmadillo, RcppEigen, gRbase

ByteCompile Yes

RoxygenNote 7.3.2

NeedsCompilation yes

Author Søren Højsgaard [aut, cre]

Repository CRAN

Date/Publication 2024-10-10 16:10:02 UTC

Contents

compileCPT 2
components_extract 3

components_gather 5
cpt 7
example_chest 8
example_grass 9
finding 9
generics 11
grain-main 12
grain-simulate 14
grain_compile 15
grain_evidence 16
grain_predict 18
grain_propagate 20
load-save-hugin 21
logical 22
mendel 24
old_components_extract 25
old_grain_evidence 26
old_replace_cpt 28
querygrain 30
repeatPattern 31
repeat_pattern 33
replace_cpt 34
simplify_query 36

Index **37**

`compileCPT` *Compile conditional probability tables / cliques potentials.*

Description

Compile conditional probability tables / cliques potentials as a preprocessing step for creating a graphical independence network

Usage

```
compileCPT(x, ..., forceCheck = TRUE)

compilePOT(x, ..., forceCheck = TRUE)
```

Arguments

`x`	To `compileCPT` x is a list of conditional probability tables; to `compilePOT`, x is a list of clique potentials.
`...`	Additional arguments; currently not used.
`forceCheck`	Controls if consistency checks of the probability tables should be made.

Details

```
* `compileCPT` is relevant for turning a collection of
cptable's into an object from which a network can be built. For
example, when specification of a cpt is made with cptable then
the levels of the node is given but not the levels of the
parents. `compileCPT` checks that the levels of variables in
the cpt's are consistent and also that the specifications
define a dag.

* `compilePOT` is not of direct relevance for the
user for the moment. However, the elements of the input should
be arrays which define a chordal undirected graph and the
arrays should, if multiplied, form a valid probability density.
```

Value

A list with a class attribute.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

extract_cpt, extract_pot, extract_marg

Examples

```
example("example_chest_cpt")
x <- compile_cpt(chest_cpt)
class(x)
grain(x)
```

---

components_extract *Extract conditional probabilities and clique potentials from data.*

Description

Extract list of conditional probability tables and list of clique potentials from data.

Usage

```
extract_cpt(data_, graph, smooth = 0)

extract_pot(data_, graph, smooth = 0)

extract_marg(data_, graph, smooth = 0)

marg2pot(marg_rep)

pot2marg(pot_rep)
```

Arguments

data_	A named array or a dataframe.
graph	An igraph object or a list or formula which can be turned into a igraph object by calling ug or dag. For extract_cpt, graph must be/define a DAG while for extract_pot, graph must be/define undirected triangulated graph.
smooth	See 'details' below.
marg_rep	An object of class marg_rep
pot_rep	An object of class pot_representation

Details

If smooth is non-zero then smooth is added to all cell counts before normalization takes place.

Value

- extract_cpt: A list of conditional probability tables.
- extract_pot: A list of clique potentials.
- extract_marg: A list of clique marginals.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

compileCPT, compilePOT, grain

Examples

```
## Extract cpts / clique potentials from data and graph
# specification and create network. There are different ways:

data(lizard, package="gRbase")

# DAG: height <- species -> diam
daG <- dag(~species + height:species + diam:species, result="igraph")

# UG : [height:species][diam:species]
uG  <- ug(~height:species + diam:species, result="igraph")

pt <- extract_pot(lizard, ~height:species + diam:species)
cp <- extract_cpt(lizard, ~species + height:species + diam:species)

pt
cp

# Both specify the same probability distribution
tabListMult(pt) |> as.data.frame.table()
tabListMult(cp) |> as.data.frame.table()

## Not run:
# Bayesian networks can be created as
bn.uG   <- grain(pt)
bn.daG  <- grain(cp)

# The steps above are wrapped into a convenience method which
# builds a network from at graph and data.
bn.uG   <- grain(uG, data=lizard)
bn.daG  <- grain(daG, data=lizard)

## End(Not run)
```

`components_gather` *Compile conditional probability tables / cliques potentials.*

Description

Compile conditional probability tables / cliques potentials as a preprocessing step for creating a graphical independence network

Usage

```
compile_cpt(x, ..., forceCheck = TRUE)

compile_pot(x, ..., forceCheck = TRUE)

parse_cpt(xi)
```

Arguments

x	To compileCPT x is a list of conditional probability tables; to compilePOT, x is a list of clique potentials.
...	Additional arguments; currently not used.
forceCheck	Controls if consistency checks of the probability tables should be made.
xi	cpt in some representation

Details

```
* `compileCPT` is relevant for turning a collection of
cptable's into an object from which a network can be built. For
example, when specification of a cpt is made with cptable then
the levels of the node is given but not the levels of the
parents. `compileCPT` checks that the levels of variables in
the cpt's are consistent and also that the specifications
define a dag.

* `compilePOT` is not of direct relevance for the
user for the moment. However, the elements of the input should
be arrays which define a chordal undirected graph and the
arrays should, if multiplied, form a valid probability density.
```

Value

A list with a class attribute.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

extract_cpt, extract_pot, extract_marg

Examples

```
example("example_chest_cpt")
x <- compile_cpt(chest_cpt)
class(x)
grain(x)
```

cpt *Create conditional probability tables (CPTs)*

Description

Creates conditional probability tables of the form p(v|pa(v)).

Usage

```
cpt(names, levels, values, normalize = "first", smooth = 0)

cptable(vpar, levels = NULL, values = NULL, normalize = TRUE, smooth = 0)
```

Arguments

`names`	Specifications of the names in P(v|pa1,...pak). See section 'details' for information about the form of the argument.
`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`	Probabilities; recycled if necessary. Regarding the order, please see section 'details' and the examples.
`normalize`	See 'details' below.
`smooth`	Should values be smoothed, see 'Details' below.
`vpar`	node an its parents

Details

`cptable` is simply a wrapper for `cpt` and the functions can hence be used synonymously.

If `smooth` is non–zero, then this value is added to all cells **before** normalization takes place.

Regarding the form of the argument `names`: To specify $P(a|b,c)$ one may write `~a|b:c`, `~a:b:c`, `~a|b+c`, `~a+b+c` or `c("a","b","c")`. Internally, the last form is used. Notice that the `+` and `:` operator are used as a separators only. The order of the variables IS important so the operators DO NOT commute.

The first variable in `levels` varies fastest.

Value

An array.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

andtable, ortable, extract_cpt, compileCPT, extract_cpt, compilePOT, grain

Examples

```
## See the wet grass example at
## https://en.wikipedia.org/wiki/Bayesian_network

yn <- c("yes", "no")
ssp <- list(R=yn, S=yn, G=yn) # state space

## Different forms
t1 <- cpt(c("S", "R"), levels=ssp,     values=c(.01, .99, .4, .6))
t2 <- cpt(~S:R,        levels=ssp,     values=c(.01, .99, .4, .6))
t3 <- cpt(~S:R,        levels=c(2, 2), values=c(.01, .99, .4, .6))
t4 <- cpt(~S:R,        levels=yn,      values=c(.01, .99, .4, .6))
t1; t2; t3; t4

varNames(t1)
valueLabels(t1)

## Wet grass example
ssp <- list(R=yn, S=yn, G=yn) # state space
p.R    <- cpt(~R,     levels=ssp, values=c(.2, .8))
p.S_R  <- cpt(~S:R,   levels=ssp, values=c(.01, .99, .4, .6))
p.G_SR <- cpt(~G:S:R, levels=ssp, values=c(.99, .01, .8, .2, .9, .1, 0, 1))

wet.cpt <- compileCPT(p.R, p.S_R, p.G_SR)
wet.cpt
wet.cpt$S # etc

# A Bayesian network is created with:
wet.bn <- grain(wet.cpt)
```

---

example_chest *Chest clinic example*

Description

Conditional probability tables for the chest clinic example.

Examples

```
yn   <- c("yes", "no")
a    <- cpt(~asia, values=c(1,99),levels=yn)
t.a  <- cpt(~tub|asia, values=c(5,95,1,99),levels=yn)
s    <- cpt(~smoke, values=c(5,5), levels=yn)
l.s  <- cpt(~lung|smoke, values=c(1,9,1,99), levels=yn)
b.s  <- cpt(~bronc|smoke, values=c(6,4,3,7), levels=yn)
e.lt <- cpt(~either|lung:tub,values=c(1,0,1,0,1,0,0,1),levels=yn)
x.e  <- cpt(~xray|either, values=c(98,2,5,95), levels=yn)
d.be <- cpt(~dysp|bronc:either, values=c(9,1,7,3,8,2,1,9), levels=yn)

chest_cpt <- list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)
## bn <- grain(compile_cpt(chest_cpt))
```

---

example_grass *Wet grass example*

Description

Conditional probability tables for the wet grass example.

Examples

```
yn <- c("yes", "no")
p.R    <- cpt(~R, values=c(.2, .8), levels=yn)
p.S_R  <- cpt(~S:R, values=c(.01, .99, .4, .6), levels=yn)
p.G_SR <- cpt(~G:S:R, values=c(.99, .01, .8, .2, .9, .1, 0, 1), levels=yn)

grass_cpt <- list(p.R, p.S_R, p.G_SR)
## bn <- grain(compile_cpt(grass_cpt))
```

---

finding *Set, retrieve, and retract finding in Bayesian network.*

Description

Set, retrieve, and retract finding in Bayesian network. NOTICE: The functions described here are kept only for backward compatibility; please use the corresponding evidence-functions in the future.

Usage

```
setFinding(object, nodes = NULL, states = NULL, flist = NULL, propagate = TRUE)
```

Arguments

object	A "grain" object
nodes	A vector of nodes
states	A vector of states (of the nodes given by 'nodes')
flist	An alternative way of specifying findings, see examples below.
propagate	Should the network be propagated?

Note

NOTICE: The functions described here are kept only for backward compatibility; please use the corresponding evidence-functions in the future:

`setEvidence()` is an improvement of `setFinding()` (and as such `setFinding` is obsolete). Users are recommended to use `setEvidence()` in the future.

`setEvidence()` allows to specification of "hard evidence" (specific values for variables) and likelihood evidence (also known as virtual evidence) for variables.

The syntax of `setEvidence()` may change in the future.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

setEvidence, getEvidence, retractEvidence, pEvidence, querygrain

Examples

```
## setFindings
yn <- c("yes", "no")
a    <- cpt(~asia, values=c(1, 99),levels=yn)
t.a  <- cpt(~tub+asia, values=c(5, 95, 1, 99),levels=yn)
s    <- cpt(~smoke, values=c(5,5), levels=yn)
l.s  <- cpt(~lung+smoke, values=c(1, 9, 1, 99), levels=yn)
b.s  <- cpt(~bronc+smoke, values=c(6, 4, 3, 7), levels=yn)
e.lt <- cpt(~either+lung+tub,values=c(1, 0, 1, 0, 1, 0, 0, 1),levels=yn)
x.e  <- cpt(~xray+either, values=c(98, 2, 5, 95), levels=yn)
d.be <- cpt(~dysp+bronc+either, values=c(9, 1, 7, 3, 8, 2, 1, 9), levels=yn)
chest.cpt <- compileCPT(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)
chest.bn <- grain(chest.cpt)

## These two forms are equivalent
bn1 <- setFinding(chest.bn, nodes=c("chest", "xray"), states=c("yes", "yes"))
bn2 <- setFinding(chest.bn, flist=list(c("chest", "yes"), c("xray", "yes")))
```

```
getFinding(bn1)
getFinding(bn2)

pFinding(bn1)
pFinding(bn2)

bn1 <- retractFinding(bn1, nodes="asia")
bn2 <- retractFinding(bn2, nodes="asia")

getFinding(bn1)
getFinding(bn2)

pFinding(bn1)
pFinding(bn2)
```

---

generics *gRain generics*

Description

Generic functions etc for the gRain package

Usage

```
nodeNames(object)

## S3 method for class 'grain'
nodeNames(object)

nodeStates(object, nodes = nodeNames(object))

## S3 method for class 'grain'
nodeStates(object, nodes = nodeNames(object))

universe(object, ...)

## S3 method for class 'grain'
universe(object, ...)

isCompiled(object)

isPropagated(object)

## S3 method for class 'cpt_spec'
vpar(object, ...)
```

```
## S3 method for class 'cpt_grain'
vpar(object, ...)

## S3 method for class 'grain'
rip(object, ...)
```

Arguments

object	A relevant object.
nodes	Some nodes of the object.
...	Additional arguments; currently not used.

---

grain-main *Create Bayesian network*

Description

Create Bayesian network (grain objects (graphical independence network)).

Usage

```
grain(x, ...)

## S3 method for class 'cpt_spec'
grain(x, control = list(), smooth = 0, compile = TRUE, details = 0, ...)

## S3 method for class 'CPTspec'
grain(x, control = list(), smooth = 0, compile = TRUE, details = 0, ...)

## S3 method for class 'pot_spec'
grain(x, control = list(), smooth = 0, compile = TRUE, details = 0, ...)

## S3 method for class 'igraph'
grain(
  x,
  control = list(),
  smooth = 0,
  compile = TRUE,
  details = 0,
  data = NULL,
  ...
)

## S3 method for class 'dModel'
grain(
```

```
  x,
  control = list(),
  smooth = 0,
  compile = TRUE,
  details = 0,
  data = NULL,
  ...
)
```

Arguments

x	An argument to build an independence network from. Typically a list of conditional probability tables, a DAG or an undirected graph. In the two latter cases, data must also be provided.
...	Additional arguments, currently not used.
control	A list defining controls, see 'details' below.
smooth	A (usually small) number to add to the counts of a table if the grain is built from a graph plus a dataset.
compile	Should network be compiled.
details	Debugging information.
data	An optional data set (currently must be an array/table)

Details

If 'smooth' is non-zero then entries of 'values' which a zero are replaced by the value of 'smooth' - BEFORE any normalization takes place.

Value

An object of class "grain"

Note

A change from earlier versions of this package is that grain objects are now compiled upon creation.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

cptable, compile.grain, propagate.grain, setFinding, setEvidence, getFinding, pFinding, retractFinding, extract_cpt, extract_pot, compileCPT, compilePOT

Examples

```
## Create network from conditional probability tables CPTs:

yn   <- c("yes", "no")
a    <- cpt(~asia,                 values=c(1,99), levels=yn)
t.a  <- cpt(~tub + asia,           values=c(5,95,1,99), levels=yn)
s    <- cpt(~smoke,                values=c(5,5), levels=yn)
l.s  <- cpt(~lung + smoke,         values=c(1,9,1,99), levels=yn)
b.s  <- cpt(~bronc + smoke,        values=c(6,4,3,7), levels=yn)
e.lt <- cpt(~either + lung + tub,  values=c(1,0,1,0,1,0,0,1), levels=yn)
x.e  <- cpt(~xray + either,        values=c(98,2,5,95), levels=yn)
d.be <- cpt(~dysp + bronc + either, values=c(9,1,7,3,8,2,1,9), levels=yn)
cpt_list  <- list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)
chest_cpt <- compileCPT(cpt_list)
## Alternative: chest_cpt <- compileCPT(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)

chest_bn  <- grain(chest_cpt)

## Create network from data and graph specification.

data(lizard, package="gRbase")

## From a DAG: height <- species -> diam
daG <- dag(~species + height:species + diam:species)

## From an undirected graph UG : [height:species][diam:species]
uG  <- ug(~height:species + diam:species)

liz_ug   <- grain(uG, data=lizard)
liz_dag  <- grain(daG, data=lizard)
```

`grain-simulate` *Simulate from Bayesian network*

Description

Simulate data from an independence network.

Usage

```
## S3 method for class 'grain'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

`object`	An independence network.
`nsim`	Number of cases to simulate.
`seed`	An optional integer controlling the random number generation.
`...`	Not used.

Value

A data frame

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

Examples

```
tf <- system.file("huginex", "chest_clinic.net", package = "gRain")

chest <- loadHuginNet(tf, details=1)
simulate(chest,n=10)

chest2 <- setFinding(chest, c("VisitToAsia", "Dyspnoea"),
                            c("yes", "yes"))
simulate(chest2, n=10)
```

---

grain_compile *Compile Bayesian network.*

Description

Compiles a Bayesian network. This means creating a junction tree and establishing clique potentials.

Usage

```
## S3 method for class 'grain'
compile(
  object,
  propagate = FALSE,
  tug = NULL,
  root = NULL,
  control = object$control,
  details = 0,
  ...
)
```

Arguments

`object`	A grain object.
`propagate`	If TRUE the network is also propagated meaning that the cliques of the junction tree are calibrated to each other.
`tug`	A triangulated undirected graph.
`root`	A set of variables which must be in the root of the junction tree
`control`	Controlling the compilation process.
`details`	For debugging info. Do not use.
`...`	Currently not used.

Value

A compiled Bayesian network; an object of class `grain`.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

`grain`, `propagate`, `propagate.grain`, `triangulate`, `rip`, `junctionTree`

`grain_evidence` *Set, update and remove evidence.*

Description

Set, update and remove evidence.

Usage

```
evidence_add(object, evidence, propagate = TRUE, details = 0)

evidence_get(object, short = TRUE)

evidence_drop(object, nodes = NULL, propagate = TRUE)

evidence_prob(object, evidence = NULL)
```

Arguments

`object`	A "grain" object
`evidence`	A list of name=value. See examples below.
`propagate`	Should the network be propagated?
`details`	Debugging information
`short`	If TRUE a dataframe with a summary is returned; otherwise a list with all details.
`nodes`	A vector of nodes.

Value

A list of tables with potentials.

Note

`setEvidence()` is an improvement of `setFinding()` (and as such `setFinding` is obsolete). Users are recommended to use `setEvidence()` in the future.

`setEvidence()` allows to specification of "hard evidence" (specific values for variables) and likelihood evidence (also known as virtual evidence) for variables.

The syntax of `setEvidence()` may change in the future.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

`setFinding`, `getFinding`, `retractFinding`, `pFinding`

Examples

```
example("grain")
chest_bn <- grain(compileCPT(chest_cpt))

bn2 <- chest_bn |> evidence_add(list(asia="yes", xray="yes"))
bn3 <- chest_bn |> evidence_add(list(asia=c(0.8, 0.1), xray="yes"))

bn2 |> evidence_get()
bn3 |> evidence_get()

bn2 |> evidence_prob()
bn3 |> evidence_prob()

bn2 |> evidence_drop("xray")
```

```
bn3 |> evidence_drop("xray")

bn2 |> evidence_drop("xray") |> evidence_get()
bn3 |> evidence_drop("xray") |> evidence_get()


## For backward compatibility these functions are available now but
# may be deprecated later.
bb2 <- setEvidence(chest_bn, c("asia", "xray"), c("yes", "yes"))
bb3 <- setEvidence(chest_bn, c("asia", "xray"), list(c(0.8, 0.2), "yes"))
bb4 <- setFinding(chest_bn, c("asia", "xray"), c("yes", "yes"))

bb2 |> getEvidence()
bb3 |> getEvidence()

bb2 |> retractEvidence("xray")
bb3 |> retractEvidence("xray")

bb2 |> pEvidence()
bb3 |> pEvidence()

bb2 |> retractEvidence("xray") |> getEvidence()
bb3 |> retractEvidence("xray") |> getEvidence()
```

`grain_predict` *Make predictions from Bayesian network*

Description

Makes predictions (either as the most likely state or as the conditional distributions) of variables conditional on finding (evidence) on other variables in an independence network.

Usage

```
## S3 method for class 'grain'
predict(
  object,
  response,
  predictors = setdiff(names(newdata), response),
  newdata,
  type = "class",
  ...
)
```

Arguments

`object`	A grain object
`response`	A vector of response variables to make predictions on

predictors	A vector of predictor variables to make predictions from. Defaults to all variables that are note responses.
newdata	A data frame
type	If "class", the most probable class is returned; if "distribution" the conditional distribution is returned.
...	Not used

Value

A list with components

pred	A list with the predictions
pFinding	A vector with the probability of the finding (evidence) on which the prediction is based

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain

Examples

```
example("example_chest_cpt")
data(chestSim500)

chest.bn <- grain(compileCPT(chest_cpt))
nd <- chestSim500[1:4]

predict(chest.bn, response="bronc", newdata=nd)
predict(chest.bn, response="bronc", newdata=nd, type="distribution")
```

grain_propagate *Propagate in a Bayesian network*

Description

Propagation refers to calibrating the cliques of the junction tree so that the clique potentials are consistent on their intersections; refer to the reference below for details.

Usage

```
## S3 method for class 'grain'
propagate(object, details = object$details, engine = "cpp", ...)

propagateLS(cq_pot_list, rip, initialize = TRUE, details = 0)
```

Arguments

object	A grain object
details	For debugging info
engine	Either "R" or "cpp"; "cpp" is the default and the fastest.
...	Currently not used
cq_pot_list	List of clique potentials
rip	A rip ordering
initialize	Always true.

Details

The propagate method invokes propagateLS which is a pure R implementation of the Lauritzen-Spiegelhalter algorithm. The c++ based version is several times faster than the purely R based version.

Value

A compiled and propagated grain object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain, compile

Examples

```
example("grain")

## Uncompiled and unpropageted network:
bn0  <- grain(chest_cpt, compile=FALSE)
bn0
## Compiled but unpropageted network:
bn1  <- compile(bn0, propagate=FALSE)
## Compiled and propagated network
bn2  <- propagate(bn1)
bn2
## Default is that networks are compiled but not propagated at creation time:
bn3  <- grain(chest_cpt)
bn3
```

---

load-save-hugin *Load and save Hugin net files*

Description

These functions can load a net file saved in the 'Hugin format' into R and save a network in R as a file in the 'Hugin format'.

Usage

```
loadHuginNet(file, description = NULL, details = 0)

saveHuginNet(gin, file, details = 0)
```

Arguments

`file`	Name of Hugin net file. Convenient to give the file the extension '.net'
`description`	A text describing the network, defaults to `file`
`details`	Debugging information.
`gin`	An independence network

Value

An object of class `grain`.

Note

- In Hugin, it is possible to specify the potential of a node as a functional relation between other nodes. In a .net file, such a specification will appear as 'function' rather than as 'node'. Such a specification is not recognized by `loadHuginNet`.
- It is recommended to avoid the text `node` as part of the name of a node.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain

Examples

```
## Load HUGIN net file
tf <- system.file("huginex", "chest_clinic.net", package = "gRain")
chest <- loadHuginNet(tf, details=1)
chest

## Save a copy
td <- tempdir()
saveHuginNet(chest, paste(td,"/chest.net",sep=''))

## Load the copy
chest2 <- loadHuginNet(paste(td,"/chest.net",sep=''))

tf <- system.file("huginex", "golf.net", package = "gRain")
golf <- loadHuginNet(tf, details=1)

saveHuginNet(golf, paste(td,"/golf.net",sep=''))
golf2 <- loadHuginNet(paste(td,"/golf.net",sep=''))
```

---

logical — *Conditional probability tables based on logical dependencies*

Description

Generate conditional probability tables based on the logical expressions AND and OR.

Usage

```
booltab(vpa, levels = c(TRUE, FALSE), op = `&`)

andtab(vpa, levels = c(TRUE, FALSE))

ortab(vpa, levels = c(TRUE, FALSE))
```

```
andtable(vpa, levels = c(TRUE, FALSE))

ortable(vpa, levels = c(TRUE, FALSE))
```

Arguments

vpa	Node and two parents; as a formula or a character vector.
levels	The levels (or rather labels) of v, see 'examples' below.
op	A logical operator.

Details

Regarding the form of the argument vpa: To specify $P(a|b,c)$ one may write ~a|b+c or ~a+b+c or ~a|b:c or ~a:b:c or c("a","b","c"). Internally, the last form is used. Notice that the + and : operator are used as separators only. The order of the variables is important so + and : DO NOT commute.

Value

An array.

Note

andtable and ortable are aliases for andtab and ortab and are kept for backward compatibility.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

cptable

Examples

```
## Logical OR:

## A variable v is TRUE if either of its parents pa1 and pa2 are TRUE:
ortab( c("v", "pa1", "pa2") ) |> ftable(row.vars="v")
## TRUE and FALSE can be recoded to e.g. yes and no:
ortab( c("v", "pa1", "pa2"), levels=c("yes", "no") ) |> ftable(row.vars="v")

## Logical AND:

## Same story here:
andtab(c("v", "pa1", "pa2") ) |> ftable(row.vars="v")
```

```
andtab(c("v", "pa1", "pa2"), levels=c("yes", "no") ) |> ftable(row.vars="v")

## Combined approach

booltab(c("v", "pa1", "pa2"), op=`&`) |> ftable(row.vars="v") ## AND
booltab(c("v", "pa1", "pa2"), op=`|`) |> ftable(row.vars="v") ## OR

booltab(~v + pa1 + pa2, op=`&`) |> ftable(row.vars="v") ## AND
booltab(~v + pa1 + pa2, op=`|`) |> ftable(row.vars="v") ## OR
```

`mendel` *Mendelian segregation*

Description

Generate conditional probability table for Mendelian segregation.

Usage

```
mendel(allele, names = c("child", "father", "mother"))
```

Arguments

`allele`	A character vector.
`names`	Names of columns in dataframe.

Note

No error checking at all on the input.

Examples

```
## Inheritance of the alleles "y" and "g"

men <- mendel(c("y","g"), names=c("ch", "fa", "mo"))
men
```

old_components_extract

Extract conditional probabilities and clique potentials from data.

Description

Extract list of conditional probability tables and list of clique potentials from data.

Usage

```
extractCPT(data_, graph, smooth = 0)

extractPOT(data_, graph, smooth = 0)

extractMARG(data_, graph, smooth = 0)
```

Arguments

data_	A named array or a dataframe.
graph	An igraph object or a list or formula which can be turned into a igraph object by calling ug or dag. For extract_cpt, graph must be/define a DAG while for extract_pot, graph must be/define undirected triangulated graph.
smooth	See 'details' below.

Details

If smooth is non-zero then smooth is added to all cell counts before normalization takes place.

Value

- extract_cpt: A list of conditional probability tables.
- extract_pot: A list of clique potentials.
- extract_marg: A list of clique marginals.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

compileCPT, compilePOT, grain

Examples

```
## Extract cpts / clique potentials from data and graph
# specification and create network. There are different ways:

data(lizard, package="gRbase")

# DAG: height <- species -> diam
daG <- dag(~species + height:species + diam:species, result="igraph")

# UG : [height:species][diam:species]
uG  <- ug(~height:species + diam:species, result="igraph")

pt <- extract_pot(lizard, ~height:species + diam:species)
cp <- extract_cpt(lizard, ~species + height:species + diam:species)

pt
cp

# Both specify the same probability distribution
tabListMult(pt) |> as.data.frame.table()
tabListMult(cp) |> as.data.frame.table()

## Not run:
# Bayesian networks can be created as
bn.uG   <- grain(pt)
bn.daG  <- grain(cp)

# The steps above are wrapped into a convenience method which
# builds a network from at graph and data.
bn.uG   <- grain(uG, data=lizard)
bn.daG  <- grain(daG, data=lizard)

## End(Not run)
```

`old_grain_evidence` *Set, update and remove evidence.*

Description

Set, update and remove evidence.

Usage

```
setEvidence(
  object,
  nodes = NULL,
  states = NULL,
  evidence = NULL,
```

```
  propagate = TRUE,
  details = 0
)

retractEvidence(object, nodes = NULL, propagate = TRUE)

absorbEvidence(object, propagate = TRUE)

getEvidence(object, short = TRUE)

pEvidence(object, evidence = NULL)
```

Arguments

object	A "grain" object
nodes	A vector of nodes.
states	A vector of states (of the nodes given by 'nodes'). Now deprecated; use argument 'evidence' instead.
evidence	A list of name=value. See examples below.
propagate	Should the network be propagated?
details	Debugging information
short	If TRUE a dataframe with a summary is returned; otherwise a list with all details.

Value

A list of tables with potentials.

Note

`setEvidence()` is an improvement of `setFinding()` (and as such `setFinding` is obsolete). Users are recommended to use `setEvidence()` in the future.

`setEvidence()` allows to specification of "hard evidence" (specific values for variables) and likelihood evidence (also known as virtual evidence) for variables.

The syntax of `setEvidence()` may change in the future.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

setFinding, getFinding, retractFinding, pFinding

Examples

```
example("grain")
chest_bn <- grain(compileCPT(chest_cpt))

bn2 <- chest_bn |> evidence_add(list(asia="yes", xray="yes"))
bn3 <- chest_bn |> evidence_add(list(asia=c(0.8, 0.1), xray="yes"))

bn2 |> evidence_get()
bn3 |> evidence_get()

bn2 |> evidence_prob()
bn3 |> evidence_prob()

bn2 |> evidence_drop("xray")
bn3 |> evidence_drop("xray")

bn2 |> evidence_drop("xray") |> evidence_get()
bn3 |> evidence_drop("xray") |> evidence_get()


## For backward compatibility these functions are available now but
# may be deprecated later.
bb2 <- setEvidence(chest_bn, c("asia", "xray"), c("yes", "yes"))
bb3 <- setEvidence(chest_bn, c("asia", "xray"), list(c(0.8, 0.2), "yes"))
bb4 <- setFinding(chest_bn, c("asia", "xray"), c("yes", "yes"))

bb2 |> getEvidence()
bb3 |> getEvidence()

bb2 |> retractEvidence("xray")
bb3 |> retractEvidence("xray")

bb2 |> pEvidence()
bb3 |> pEvidence()

bb2 |> retractEvidence("xray") |> getEvidence()
bb3 |> retractEvidence("xray") |> getEvidence()
```

`old_replace_cpt` *Replace CPTs in Bayesian network*

Description

Replace CPTs of Bayesian network.

Usage

```
replaceCPT(object, value)
```

Arguments

`object`	A `grain` object.
`value`	A named list, see examples below.

Details

When a Bayesian network (BN) is constructed from a list of conditional probability tables (CPTs) (e.g. using the function `grain()`), various actions are taken:

1. It is checked that the list of CPTs define a directed acyclic graph (DAG).
2. The DAG is moralized and triangulated.
3. A list of clique potentials (one for each clique in the triangulated graph) is created from the list of CPTs.
4. The clique potentials are, by default, calibrated to each other so that the potentials contain marginal distributions.

The function described here bypass the first two steps which can provide an important gain in speed compared to constructing a new BN with a new set of CPTs with the same DAG.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain, propagate, triangulate, rip, junctionTree

Examples

```
## See the wet grass example at
## https://en.wikipedia.org/wiki/Bayesian_network

yn <- c("yes", "no")
p.R    <- cptable(~R, values=c(.2, .8), levels=yn)
p.S_R  <- cptable(~S:R, values=c(.01, .99, .4, .6), levels=yn)
p.G_SR <- cptable(~G:S:R, values=c(.99, .01, .8, .2, .9, .1, 0, 1), levels=yn)

wet.bn <- compileCPT(p.R, p.S_R, p.G_SR)  |> grain()
getgrain(wet.bn, "cpt")[c("R","S")]

# Update some CPTs
wet.bn <- replace_cpt(wet.bn, list(R=c(.3, .7), S=c(.1, .9, .7, .3)))
getgrain(wet.bn, "cpt")[c("R","S")]
```

querygrain *Query a Bayesian network*

Description

Query an independence network, i.e. obtain the conditional distribution of a set of variables - possibly (and typically) given finding (evidence) on other variables.

Usage

```
querygrain(
  object,
  nodes = nodeNames(object),
  type = "marginal",
  evidence = NULL,
  exclude = TRUE,
  normalize = TRUE,
  simplify = FALSE,
  result = "array",
  details = 0
)
```

Arguments

object	A grain object.
nodes	A vector of nodes; those nodes for which the (conditional) distribution is requested.
type	Valid choices are "marginal" which gives the marginal distribution for each node in nodes; "joint" which gives the joint distribution for nodes and "conditional" which gives the conditional distribution for the first variable in nodes given the other variables in nodes.
evidence	An alternative way of specifying findings (evidence), see examples below.
exclude	If TRUE then nodes on which evidence is given will be excluded from nodes (see above).
normalize	Should the results be normalized to sum to one.
simplify	Should the result be simplified (to a dataframe) if possible.
result	If "data.frame" the result is returned as a data frame (or possibly as a list of dataframes).
details	Debugging information

Value

A list of tables with potentials.

Note

setEvidence() is an improvement of setFinding() (and as such setFinding is obsolete). Users are recommended to use setEvidence() in the future.

setEvidence() allows to specification of "hard evidence" (specific values for variables) and likelihood evidence (also known as virtual evidence) for variables.

The syntax of setEvidence() may change in the future.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

setEvidence, getEvidence, retractEvidence, pEvidence

Examples

```
testfile <- system.file("huginex", "chest_clinic.net", package = "gRain")
chest <- loadHuginNet(testfile, details=0)
qb <- querygrain(chest)
qb

lapply(qb, as.numeric) # Safe
sapply(qb, as.numeric) # Risky
```

---

repeatPattern *Create repeated patterns in Bayesian networks*

Description

Repeated patterns is a useful model specification short cut for Bayesian networks

Usage

```
repeatPattern(plist, instances, unlist = TRUE, data = NULL)
```

Arguments

plist	A list of conditional probability tables. The variable names must have the form name[i] and the i will be substituted by the values given in instances below. See also the data argument.
instances	A vector of consecutive integers
unlist	If FALSE the result is a list in which each element is a copy of plist in which name[i] are substituted. If TRUE the result is the result of applying unlist().
data	A two column matrix. The first column is the index / name of a node; the second column is the index / name of the node's parent.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain, compile_cpt

Examples

```
yn <- c("yes", "no")
n <- 3

## Example: Markov chain

x_init  <- cpt(~x0, values=c(1, 9), levels=yn)                  ## p(x0)
x_trans <- cpt(~x[i]|x[i-1], values=c(1, 99, 2, 98), levels=yn) ## p(x[i]|x[i-1])
pat     <- list(x_trans)
rep.pat <- repeat_pattern(pat, instances=1:n)

mc <- compile_cpt(c(list(x_init), rep.pat))
mc
mc <- mc |> grain()

## Example: Hidden markov model:
# The x[i]'s are unobserved, the y[i]'s can be observed.

x_init  <- cpt(~x0, values=c(1, 9), levels=yn)                   ##  p(x0)
x_trans <- cpt(~x[i]|x[i-1], values=c(1, 99, 2, 98), levels=yn)  ##  p(x[i]|x[i-1])
y_emis  <- cpt(~y[i]|x[i], values=c(10, 90, 20, 80), levels=yn)  ##  p(y[i]|x[i])

pat     <- list(x_trans, y_emis) ## Pattern to be repeated
rep.pat <- repeat_pattern(pat, instances=1:n)
hmm <- compile_cpt(c(list(x_init), rep.pat))
hmm
```

```
hmm <- hmm |> grain()

## Data-driven variable names

dep <- data.frame(i=c(1, 2, 3, 4, 5, 6, 7, 8),
                  p=c(0, 1, 2, 2, 3, 3, 4, 4))

x0 <- cpt(~x0, values=c(0.5, 0.5), levels=yn)
xa <- cpt(~x[i] | x[data[i, "p"]], values=c(1, 9, 2, 8), levels=yn)
xb <- repeat_pattern(list(xa), instances=1:nrow(dep), data=dep)
tree <- compile_cpt(c(list(x0), xb))
tree
tree <- tree |> grain()
tree
```

`repeat_pattern` *Create repeated patterns in Bayesian networks*

Description

Repeated patterns is a useful model specification short cut for Bayesian networks

Usage

```
repeat_pattern(plist, instances, unlist = TRUE, data = NULL)
```

Arguments

`plist`	A list of conditional probability tables. The variable names must have the form `name[i]` and the `i` will be substituted by the values given in `instances` below. See also the `data` argument.
`instances`	A vector of consecutive integers
`unlist`	If `FALSE` the result is a list in which each element is a copy of `plist` in which `name[i]` are substituted. If `TRUE` the result is the result of applying `unlist()`.
`data`	A two column matrix. The first column is the index / name of a node; the second column is the index / name of the node's parent.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

grain, compile_cpt

Examples

```
yn <- c("yes", "no")
n <- 3

## Example: Markov chain

x_init  <- cpt(~x0, values=c(1, 9), levels=yn)                  ## p(x0)
x_trans <- cpt(~x[i]|x[i-1], values=c(1, 99, 2, 98), levels=yn) ## p(x[i]|x[i-1])
pat     <- list(x_trans)
rep.pat <- repeat_pattern(pat, instances=1:n)

mc <- compile_cpt(c(list(x_init), rep.pat))
mc
mc <- mc |> grain()

## Example: Hidden markov model:
# The x[i]'s are unobserved, the y[i]'s can be observed.

x_init  <- cpt(~x0, values=c(1, 9), levels=yn)                   ##  p(x0)
x_trans <- cpt(~x[i]|x[i-1], values=c(1, 99, 2, 98), levels=yn)  ##  p(x[i]|x[i-1])
y_emis  <- cpt(~y[i]|x[i], values=c(10, 90, 20, 80), levels=yn)  ##  p(y[i]|x[i])

pat     <- list(x_trans, y_emis) ## Pattern to be repeated
rep.pat <- repeat_pattern(pat, instances=1:n)
hmm <- compile_cpt(c(list(x_init), rep.pat))
hmm
hmm <- hmm |> grain()

## Data-driven variable names

dep <- data.frame(i=c(1, 2, 3, 4, 5, 6, 7, 8),
                  p=c(0, 1, 2, 2, 3, 3, 4, 4))

x0 <- cpt(~x0, values=c(0.5, 0.5), levels=yn)
xa <- cpt(~x[i] | x[data[i, "p"]], values=c(1, 9, 2, 8), levels=yn)
xb <- repeat_pattern(list(xa), instances=1:nrow(dep), data=dep)
tree <- compile_cpt(c(list(x0), xb))
tree
tree <- tree |> grain()
tree
```

---

replace_cpt *Replace CPTs in Bayesian network*

Description

Replace CPTs of Bayesian network.

Usage

```
replace_cpt(object, value)

## S3 method for class 'cpt_grain'
replace_cpt(object, value)
```

Arguments

`object`	A `grain` object.
`value`	A named list, see examples below.

Details

When a Bayesian network (BN) is constructed from a list of conditional probability tables (CPTs) (e.g. using the function `grain()`), various actions are taken:

1. It is checked that the list of CPTs define a directed acyclic graph (DAG).
2. The DAG is moralized and triangulated.
3. A list of clique potentials (one for each clique in the triangulated graph) is created from the list of CPTs.
4. The clique potentials are, by default, calibrated to each other so that the potentials contain marginal distributions.

The function described here bypass the first two steps which can provide an important gain in speed compared to constructing a new BN with a new set of CPTs with the same DAG.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. https://www.jstatsoft.org/v46/i10/.

See Also

`grain`, `propagate`, `triangulate`, `rip`, `junctionTree`

Examples

```
## See the wet grass example at
## https://en.wikipedia.org/wiki/Bayesian_network

yn <- c("yes", "no")
p.R    <- cptable(~R, values=c(.2, .8), levels=yn)
p.S_R  <- cptable(~S:R, values=c(.01, .99, .4, .6), levels=yn)
p.G_SR <- cptable(~G:S:R, values=c(.99, .01, .8, .2, .9, .1, 0, 1), levels=yn)

wet.bn <- compileCPT(p.R, p.S_R, p.G_SR)  |> grain()
getgrain(wet.bn, "cpt")[c("R","S")]

# Update some CPTs
wet.bn <- replace_cpt(wet.bn, list(R=c(.3, .7), S=c(.1, .9, .7, .3)))
getgrain(wet.bn, "cpt")[c("R","S")]
```

`simplify_query` *Simplify output query to a Bayesian network*

Description

Simplify output query to a Bayesian network to a dataframe provided that each node has the same levels.

Usage

```
simplify_query(b)
```

Arguments

b	Result from running querygrain.

Index

∗ **datasets**
- example_chest, 8
- example_grass, 9

∗ **models**
- cpt, 7
- finding, 9
- grain-main, 12
- grain-simulate, 14
- grain_compile, 15
- grain_evidence, 16
- grain_predict, 18
- grain_propagate, 20
- querygrain, 30
- repeat_pattern, 33
- replace_cpt, 34

∗ **old_names**
- compileCPT, 2
- finding, 9
- old_components_extract, 25
- old_grain_evidence, 26
- old_replace_cpt, 28
- repeatPattern, 31

∗ **utilities**
- components_extract, 3
- components_gather, 5
- cpt, 7
- finding, 9
- grain_compile, 15
- grain_evidence, 16
- grain_propagate, 20
- load-save-hugin, 21
- logical, 22
- querygrain, 30
- replace_cpt, 34

absorbEvidence *(old_grain_evidence)*, 26
andtab *(logical)*, 22
andtable, *8*
andtable *(logical)*, 22
ask *(querygrain)*, 30
booltab *(logical)*, 22

compile, *20*
compile.cpt_grain *(grain_compile)*, 15
compile.grain, *13*
compile.grain *(grain_compile)*, 15
compile.pot_grain *(grain_compile)*, 15
compile_cpt, *32*, *34*
compile_cpt *(components_gather)*, 5
compile_pot *(components_gather)*, 5
compileCPT, 2, *4*, *8*, *13*, *25*
compilePOT, *4*, *8*, *13*, *25*
compilePOT *(compileCPT)*, 2
components_extract, 3
components_gather, 5
cpt, 7
cptable, *13*, *23*
cptable *(cpt)*, 7

evidence_add *(grain_evidence)*, 16
evidence_drop *(grain_evidence)*, 16
evidence_get *(grain_evidence)*, 16
evidence_prob *(grain_evidence)*, 16
example_chest, 8
example_chest_cpt *(example_chest)*, 8
example_grass, 9
example_grass_cpt *(example_grass)*, 9
extract_cpt, *3*, *6*, *8*, *13*
extract_cpt *(components_extract)*, 3
extract_marg, *3*, *6*
extract_marg *(components_extract)*, 3
extract_pot, *3*, *6*, *13*
extract_pot *(components_extract)*, 3
extractCPT *(old_components_extract)*, 25
extractMARG *(old_components_extract)*, 25
extractPOT *(old_components_extract)*, 25

finding, 9

generics, 11

getEvidence, *10*, *31*
getEvidence *(*old_grain_evidence*)*, 26
getFinding, *13*, *17*, *27*
getFinding *(*finding*)*, 9
grain, *4*, *8*, *16*, *19*, *20*, *22*, *25*, *29*, *32*, *34*, *35*
grain *(*grain-main*)*, 12
grain-main, 12
grain-simulate, 14
grain.cpt_spec *(*grain-main*)*, 12
grain.CPTspec *(*grain-main*)*, 12
grain.dModel *(*grain-main*)*, 12
grain.igraph *(*grain-main*)*, 12
grain.pot_spec *(*grain-main*)*, 12
grain_compile, 15
grain_evidence, 16
grain_predict, 18
grain_propagate, 20

isCompiled *(*generics*)*, 11
isPropagated *(*generics*)*, 11

junctionTree, *16*, *29*, *35*

load-save-hugin, 21
loadHuginNet *(*load-save-hugin*)*, 21
logical, 22

marg2pot *(*components_extract*)*, 3
mendel, 24

nodeNames *(*generics*)*, 11
nodeStates *(*generics*)*, 11

old_components_extract, 25
old_grain_evidence, 26
old_replace_cpt, 28
ortab *(*logical*)*, 22
ortable, *8*
ortable *(*logical*)*, 22

parse_cpt *(*components_gather*)*, 5
parse_cpt, *(*components_gather*)*, 5
parse_cpt.default *(*components_gather*)*, 5
parse_cpt.xtabs, *(*components_gather*)*, 5
pEvidence, *10*, *31*
pEvidence *(*old_grain_evidence*)*, 26
pFinding, *13*, *17*, *27*
pFinding *(*finding*)*, 9
pot2marg *(*components_extract*)*, 3
predict.grain *(*grain_predict*)*, 18

propagate, *16*, *29*, *35*
propagate.grain, *13*, *16*
propagate.grain *(*grain_propagate*)*, 20
propagateLS *(*grain_propagate*)*, 20
propagateLS__ *(*grain_propagate*)*, 20

qgrain *(*querygrain*)*, 30
querygrain, *10*, 30

repeat_pattern, 33
repeatPattern, 31
replace_cpt, 34
replaceCPT *(*old_replace_cpt*)*, 28
retractEvidence, *10*, *31*
retractEvidence *(*old_grain_evidence*)*, 26
retractFinding, *13*, *17*, *27*
retractFinding *(*finding*)*, 9
rip, *16*, *29*, *35*
rip.grain *(*generics*)*, 11

saveHuginNet *(*load-save-hugin*)*, 21
setEvidence, *10*, *13*, *31*
setEvidence *(*old_grain_evidence*)*, 26
setFinding, *13*, *17*, *27*
setFinding *(*finding*)*, 9
simplify_query, 36
simulate.grain *(*grain-simulate*)*, 14

triangulate, *16*, *29*, *35*

universe *(*generics*)*, 11

vpar.cpt_grain *(*generics*)*, 11
vpar.cpt_spec *(*generics*)*, 11