Package ‘gRain’

October 17, 2016

Version 1.3-0
Title Graphical Independence Networks
Author Søren Højsgaard <sorenh@math.aau.dk>
Maintainer Søren Højsgaard <sorenh@math.aau.dk>
Description Probability propagation in graphical independence networks, also
known as Bayesian networks or probabilistic expert systems.
License GPL (>= 2)
Depends R (>= 3.0.2), methods, gRbase (>= 1.7-2)
Imports igraph, graph, magrittr, functional, Rcpp (>= 0.11.1)
URL http://people.math.aau.dk/~sorenh/software/gR/
Encoding UTF-8
Suggests Rgraphviz, microbenchmark
LinkingTo Rcpp (>= 0.11.1), RcppArmadillo, RcppEigen, gRbase (>=
1.8-0)
ByteCompile Yes
RoxygenNote 5.0.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2016-10-17 11:09:28

R topics documented:

compile-cpt ................................................................. 2
cptable ................................................................. 3
evidence-object ......................................................... 4
extract-cpt ............................................................... 6
finding ................................................................. 8
grain-compile ......................................................... 10
grain-evi ............................................................. 11
grain-evidence ......................................................... 14
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, details = 0)
```

```
compilePOT(x)
```

Arguments

- **x**: To compileCPT x is a list of conditional probability tables; to compilePOT, x is a list of clique potentials.
- **forceCheck**: Controls if consistency checks of the probability tables should be made.
- **details**: Controls amount of print out. Mainly for debugging purposes.

Value

- `compileCPT` returns a list of class `cptspec`
- `compilePOT` returns a list of class `potspec`

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References

**cptable**

**See Also**

extractCPT, extractPOT

**Description**

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

**Usage**

\[
\text{cptable}(vpar, \text{ levels } = \text{ NULL}, \text{ values } = \text{ NULL}, \text{ normalize } = \text{ TRUE}, \\
\text{ smooth } = \emptyset)
\]

**Arguments**

- **vpar**: Specifications of the names in \( P(v|pa1,...,pak) \). See section 'details' for information about the form of the argument.
- **levels**: See 'details' below.
- **values**: Probabilities; recycled if necessary. Regarding the order, please see section 'details' and the examples.
- **normalize**: See 'details' below.
- **smooth**: See 'details' below.

**Details**

If \text{normalize} = \text{TRUE} then for each configuration of the parents the probabilities are normalized to sum to one.

If \text{smooth} is non–zero then zero entries of \text{values} are replaced with \text{smooth} before normalization takes place.

Regarding the form of the argument \text{vpar}: To specify \( P(a|b,c) \) one may write \( \neg a|b:c, \neg a:b:c, \neg a|b+c, \neg a+b+c \) or \( c("a","b","c") \). Internally, the last form is used. Notice that the + and : operator is used as a separator only. The order of the variables is important so the operators do not commute.

If \( a \) has levels \( a1, a2 \) and likewise for \( b \) and \( c \) then the order of \text{values} corresponds to the configurations \( (a1,b1,c1), (a2,b1,c1), (a1,b2,c1), (a2,b2,c1) \) etc. That is, the first variable varies fastest. Hence the first two elements in \text{values} will be the conditional probabilities of a given \( b=b1, c=c1 \).

**Value**

A cptable object (a list).
Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>

References

See Also
andtable, ortable, extractCPT, compileCPT, extractPOT, compilePOT, grain

Examples

yn <- c("yes", "no")
ynm <- c("yes", "no", "maybe")
a <- cptable(~ asia, values=c(1,99), levels=yn)
t.a <- cptable(~ tub : asia, values=c(5,95,1,99,1,999), levels=ynm)
d.a <- cptable(~ dia : asia, values=c(5,5,1,99,100,999), levels=ynm)
cptlist <- compileCPT(list(a,t.a,d.a))
grain(cptlist)

## Example: Specifying conditional probabilities as a matrix
bayes.levels <- c("Enzyme", "Keratine", "unknown")
root.node <- cptable(~R, values=c(1,1,1), levels=bayes.levels)
cond.prob.tbl <- t(matrix(c(1,0,0,0,1,0,0.5,0.5,0),
                        nrow=3, ncol=3, byrow=TRUE, dimnames=list(bayes.levels, bayes.levels)))
cond.prob.tbl

## Notice above: Columns represent parent states; rows represent child states
query.node <- cptable(~ Q | R, values=cond.prob.tbl, levels=bayes.levels)
sister.node <- cptable(~ S | R, values=cond.prob.tbl, levels=bayes.levels)

## Testing
compile(grain(compileCPT(list( root.node, query.node, sister.node ))), propagate=TRUE)

evidence-object

<table>
<thead>
<tr>
<th>evidence-object</th>
<th>Evidence objects</th>
</tr>
</thead>
</table>

Description
Functions for defining and manipulating evidence.
Usage

```r
evi.list = NULL, levels)

is.null_ev(object)

## S3 method for class 'grain.ev'
print(x, ...)

## S3 method for class 'grain.ev'
varNames(x)

## S3 method for class 'grain.ev'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

setdiff_ev(ev1, ev2)

union_ev(ev1, ev2)
```

Arguments

- `evi.list`: A named list with evidence; see 'examples' below.
- `levels`: A named list with the levels of all variables.
- `object`: Some R object.
- `x`: Evidence object
- `...`: Not used.
- `row.names`: Not used.
- `optional`: Not used.
- `ev1`, `ev2`: Evidence.

Details

Evidence is specified as a list. Internally, evidence is represented as a grain evidence object which is a list with 4 elements.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

Examples

```r
## Define the universe

uni <- list(asia = c("yes", "no"), tub = c("yes", "no"), smoke = c("yes", "no"),
             lung = c("yes", "no"), bronc = c("yes", "no"), either = c("yes", "no"),
             xray = c("yes", "no"), dysp = c("yes", "no"))
```
e1 <- list(dysp="no", xray="no")
e1 <- new_ev( e1, levels=uni )
e1
as.data.frame( e1 )
e1 %>% str
e1.2 <- list(dysp="no", xray=c(0, 1))
e1.2 <- new_ev( e1.2, levels=uni )
e1.2

# Notice that in \code{e1.2}, \code{xray} is not regarded as hard
# evidence but as a weight on each level. Other than that \code{e1.2}
# and \code{e1} are equivalent here. This is used in connection
# with specifying likelihood evidence.
e2 <- list(dysp="yes", asia="yes")
e2 <- new_ev(e2, uni)

# If evidence 'e1' is already set in the network and new evidence
# 'e2' emerges, the evidence in the network must be updated. But
# there is a conflict in that \code{dysp="yes"} in 'e1' and
# \code{dysp="no"} in 'e2'. The (arbitrary) convention is that
# existing evidence overrides new evidence so that the only new
# evidence in 'e2' is really \code{asia="yes"}.

# To subtract existing evidence from new evidence we can do:
setdiff_ev( e2, e1 )

# Likewise the 'union' is
union_ev( e2, e1 )

---

**extract-cpt**

*Extract conditional probabilities and clique potentials from data.*

**Description**

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

**Usage**

```
extractCPT(x, graph, smooth = 0)
extractPOT(x, graph, smooth = 0)
```

**Arguments**

- `x` An array or a dataframe.
graph A graph represented as a graphNEL object. For extractCPT, graph must be a DAG while for extractPOT, graph must be 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
extractCPT: A list of conditional probability tables
extractPOT: A list of clique potentials.

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

References

See Also
compileCPT, compilePOT, grain

Examples

## Asia (chest clinic) example:

### Version 1) Specify conditional probability tables.

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

### Version 2) Specify DAG and data

data(chestSim100000, package="gRbase")
dgf <- ~asia + tub * asia + smoke + lung * smoke + bronc * smoke + either + tub * lung + xray * either + dysp * bronc * either
dg <- dag(dgf)
pp <- extractCPT(chestSim100000, dg)
cpp2 <- compileCPT(pp)
pn2 <- grain/cpp2)
q2 <- querygrain(pn2)

## Version 2) Specify triangulated undirected graph and data
ugf <- list(c("either", "lung", "tub"), c("either", "lung", "bronc"),
c("either", "xray"), c("either", "dysp", "bronc"), c("smoke",
"lung", "bronc"), c("asia", "tub"))
gg <- ugList(ugf)
pp <- extractPOT(chestSim10000, gg)
cpp3 <- compilePOT(pp)
pn3 <- grain/cpp3)
q3 <- querygrain(pn3)

## Compare results:
str(q1)
str(q2[names(q1)])
str(q3[names(q1)])

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

See Also
setEvidence getEvidence retractEvidence pEvidence querygrain

Examples

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

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

gFinding(bn1)
gFinding(bn2)

pFinding(bn1)
pFinding(bn2)

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

gFinding(bn1)
gFinding(bn2)

pFinding(bn1)
pFinding(bn2)
```
grain-compile

Compile a graphical independence network (a 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, root = NULL,
        control = object$control, details = 0, ...)

## S3 method for class 'CPTgrain'
cOMPile(object, propagate = FALSE, root = NULL,
        control = object$control, details = 0, ...)

## S3 method for class 'POTgrain'
cOMPile(object, propagate = FALSE, 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.
- **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

See Also

grain, propagate, triangulate, rip, junctionTree

---

**grain-evi**  
*Set evidence in grain objects*

**Description**

Setting and removing evidence in grain objects.

**Usage**

```r
setEvi(object, nodes = NULL, states = NULL, evidence = NULL, propagate = TRUE, details = 0)
setEvi_(object, evidence = NULL, propagate = TRUE, details = 0)
retractEvi(object, items = NULL, propagate = TRUE)
retractEvi_(object, items = NULL, propagate = TRUE)
absorbEvi(object, propagate = TRUE)
absorbEvi_(object, propagate = TRUE)
pEvidence(object)
getEvidence(object)
dropEvi(object) <- value
addEvi(object) <- value
evidence(object)

## S3 method for class 'grain'
evidence(object)
evidence(object) <- value

## S3 replacement method for class 'grain'
evidence(object) <- value

addEvi(object, nodes = NULL, states = NULL, evidence = NULL, propagate = TRUE, details = 0)
```
dropEvi(object, items = NULL, propagate = TRUE)

getEvi(object)

insertEvi(evi.list, pot, hostclique)

getHostClique(set.list, cliques)

**Arguments**

- **object** A "grain" object
- **nodes** A vector of nodes; those nodes for which the (conditional) distribution is requested.
- **states** A vector of states (of the nodes given by 'nodes
- **evidence** An alternative way of specifying findings (evidence), see examples below.
- **propagate** Should the network be propagated?
- **details** Debugging information
- **items** Items in the evidence list to be removed. Here, NULL means remove everything. If items is a character vector (of nodes) then evidence on these nodes is removed. If items is a numeric vector then those items in the evidence list is removed. Notice that 0 means nothing is removed.
- **value** The evidence in the form of a named list or an evidence-object.
- **evi.list** A "grain_ev" object.
- **pot** A list of clique potentials (a potential is an array).
- **hostclique** A numerical vector indicating in which element of 'pot' each evidence item in 'evi.list' should be inserted in.
- **set.list** A list of sets (a set is a character vector).
- **cliques** A list of sets (a set is a character vector).

**Author(s)**

Søren Højsgaard, <sorenh@math.aau.dk>

**References**


**See Also**

Examples

```r
## setFinding / setEvidence

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

## 1) These forms are identical

e1 <- list(dyisp="no", xray="no")
setEvI(bn, evidence=e1)
setEvI(bn, nodes=c("dyisp","xray"), states=c("no", "no"))
setEvidence(bn, nodes=c("dyisp","xray"), states=c("no", "no"))

# Notice: setFinding is old school but it was used in the
# "Graphical Models with R" book.
setFinding(bn, nodes=c("dyisp","xray"), states=c("no", "no"))

## 2) Updating evidence
# Notice that only 'asia' is set because 'dyisp' was set earlier

e2 <- list(dyisp="yes", asia="yes")
bn1 <- setEvI(bn, evidence=e1)
bn1
bn2 <- setEvI(bn1, evidence=e2)
bn2

## 3) Shorter forms

bn2 <- bn
evidence(bn2) <- e1
evidence(bn2) <- e2

## 4) Alternative forms:

setEvI(bn, evidence=list("asia"=c(1,0), "xray"="yes"))
```
## Description

Set, update and remove evidence.

## Usage

```r
setEvidence(object, nodes = NULL, states = NULL, evidence = NULL, nslist = NULL, propagate = TRUE, details = NULL)
```

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

```r
absorbEvidence(object, propagate = TRUE)
```

## Arguments

- **object**: A "grain" object
- **nodes**: A vector of nodes; those nodes for which the (conditional) distribution is requested.
- **states**: A vector of states (of the nodes given by 'nodes')
- **evidence**: An alternative way of specifying findings (evidence), see examples below.
- **nslist**: deprecated
- **propagate**: Should the network be propagated?
- **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**


**See Also**

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

**Examples**

```r

# 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

## setFinding / setEvidence

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

## 1) These two forms are identical

setEvidence(chest, c("asia","xray"), c("yes", "yes"))
setFinding(chest, c("asia","xray"), c("yes", "yes"))

## 2) Suppose we do not know with certainty whether a patient has
```
Grain generics

Description

Generic functions etc for the gRain package

Usage

nodenames(x)

## S3 method for class 'grain'
nodenames(x)

nodestates(x, nodes = nodenames(x))

## S3 method for class 'grain'
nodestates(x, nodes = nodenames(x))

universe(object, ...)

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

## S3 method for class 'grainEvidence_
varnames(x)

Arguments

x, object A relevant object.

nodes Some nodes of the object.

... Additional arguments; currently not used.
Description

The ‘grain’ builds a graphical independence network.

Usage

grain(x, data = NULL, control = list(), smooth = 0, details = 0, ...)

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

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

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

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

is.grain(object)

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.

data An optional data set (currently must be an array/table)

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.

details Debugging information.

... Additional arguments, currently not used.

object Any R object.

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"

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

cptable, compile.grain, propagate.grain, setFinding, setEvidence, getFinding, pFinding, retractFinding

Examples

```r
## Asia (chest clinic) example:
y <- c("yes","no")
a <- cptable(~asia, values=c(1,99), levels=y)
t.a <- cptable(~tub+asia, values=c(5,95,1,99), levels=y)
s <- cptable(~smoke, values=c(5,5), levels=y)
l.s <- cptable(~lung+smoke, values=c(1,9,1,99), levels=y)
b.s <- cptable(~bronc+smoke, values=c(6,4,3,7), levels=y)
e.lt <- cptable(~either+lung+tub, values=c(1,0,1,0,1,0,0,1), levels=y)
x.e <- cptable(~xray+either, values=c(98,2,5,95), levels=y)
d.be <- cptable(~dysp+bronc+either, values=c(9,1,7,3,8,2,1,9), levels=y)
plist <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
bn <- grain(pplist)
bn
summary(bn)
plot(bn)
bnc <- compile(bn, propagate=TRUE)

## If we want to query the joint distribution of the disease nodes,
## computations can be speeded up by forcing these nodes to be in
## the same clique of the junction tree:

bnc2 <- compile(bn, root=c("lung", "bronc", "tub"), propagate=TRUE)

system.time(
  for (i in 1:200) 
    querygrain(bnc, nodes=c("lung","bronc", "tub"), type="joint"))

system.time(
  for (i in 1:200) 
    querygrain(bnc2, nodes=c("lung","bronc", "tub"), type="joint"))

## Simple example - one clique only in triangulated graph:
```
grain-predict

grain-predict: Make predictions from a probabilistic 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 not 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

See Also

grain

grain-propagate

Propagate a graphical independence network (a Bayesian network)

Description

Propagation refers to calibrating the cliques of the junction tree so that the clique potentials are
consistent on their intersections

Usage

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

propagate__h(object, details = object$details, ...)

propagateLS(cqpotList, rip, initialize = TRUE, details = 0)
Arguments

- **object**: A grain object
- **details**: For debugging info
- **...**: Currently not used
- **cqpotList**: Clique potential list
- **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 function `propagate__` invokes `propagateLS__` which is a C++ implementation of the Lauritzen-Spiegelhalter algorithm.

The C++ based version is several times faster than the purely R based version, and after some additional testing the C++ based version will become the default.

Value

A compiled and propagated grain object.

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

`grain`, `compile`

Examples

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

Simulate from an independence network

Description

Simulate data from an independence network.

Usage

```r
## 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

Examples

```r
## Not run:

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)

## End(Not run)
```

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

```r
loadHuginNet(file, description = rev(unlist(strsplit(file, "/"))[1],
            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 (a list) of class "huginNet".

Author(s)

S<8>ren H<8>jsgaard, <sorenh@math.aau.dk>
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))
logical

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 $\neg a|b+c$ or $\neg a+b+c$ or $\neg a|b:c$ or $\neg a:b:c$ 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 + does 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


See Also

cptable

Examples

```r
## 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
```
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.

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

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

querygrain

Query a 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, result = "array",
details = 0)

## S3 method for class 'grain'
querygrain(object, nodes = nodeNames(object),
type = "marginal", evidence = NULL, exclude = TRUE, normalize = TRUE,
result = "array", details = 0)
querygrain

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.

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


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

## setFinding / setEvidence

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

## 1) These two forms are identical
setEvidence(chest, c("asia", "xray"), c("yes", "yes"))
setFinding(chest, c("asia", "xray"), c("yes", "yes"))

## 2) Suppose we do not know with certainty whether a patient has
## recently been to Asia. We can then introduce a new variable
## "guess.asia" with "asia" as its only parent. Suppose
## p(guess.asia=yes|asia=yes)=.8 and p(guess.asia=yes|asia=no)=.1
## If the patient is e.g. unusually tanned we may set
## guess.asia=yes and propagate. This corresponds to modifying the
## model by the likelihood (0.8, 0.1) as
## setEvidence(chest, c("asia","xray"), list(c(0.8,0.1), "yes"))

## 3) Hence, the same result as in 1) can be obtained with
## setEvidence(chest, c("asia","xray"), list(c(1, 0), "yes"))

## 4) An alternative specification using evidence is
## setEvidence(chest, evidence=list("asia"=c(1, 0), "xray"="yes"))

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)

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.
- **instances**: A vector of distinct 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()`. 

Author(s)

Søren Højsgaard, <sorenh@math.aau.dk>

References


See Also

- `grain`, `compilecpt`

Examples

```r
## Specify hidden markov models. The x[i]'s are unobserved, the
## y[i]'s can be observed.

yn <- c("yes","no")

## Specify p(x0)
x.0 <-cptable(~x0, values=c(1,1), levels=yn)

## Specify transition density
x.x <- ctable(~x[i]|x[i-1], values=c(1,99,2,98),levels=yn)

## Specify emission density
y.x <- ctable(~y[i]|x[i], values=c(1,99,2,98),levels=yn)

## The pattern to be repeated
pp <- list(x.x, y.x)

## Repeat pattern and create network
ppp <- repeatPattern(pp, instances=1:10)
qqq <- compileCPT(c(list(x.0),ppp))
rrr <- grain(qqq)
```
\textbf{set-jevidence} \hfill \textit{Set joint evidence in grain objects}

\textbf{Description}

Setting and removing joint evidence in grain objects.

\textbf{Usage}

\begin{verbatim}
setJEv(object, evidence = NULL, propagate = TRUE, details = 0)
insertJEv(evi.list, pot, hostclique)
retractJEv(object, items = NULL, propagate = TRUE, details = 0)
new_jev(ev, levels)

## S3 method for class 'grain_jev'
print(x, ...)
\end{verbatim}

\textbf{Arguments}

- \texttt{object} A "grain" object
- \texttt{evidence} A list of evidence. Each element is a named array.
- \texttt{propagate} Should the network be propagated?
- \texttt{details} Debugging information
- \texttt{evi.list} A "grain_jev" object.
- \texttt{pot} A list of clique potentials (a potential is an array).
- \texttt{hostclique} A numerical vector indicating in which element of 'pot' each evidence item in 'evi.list' should be inserted in.
- \texttt{items} Items in the evidence list to be removed. Here, NULL means remove everything, 0 means nothing is removed. Otherwise \texttt{items} is a numeric vector.
- \texttt{ev} A named list.
- \texttt{levels} A named list.
- \texttt{x} A "grain_jev" object.
- \texttt{...} Additional arguments; currently not used.

\textbf{Note}

All the joint evidence functionality should be used \textit{with great care}.

\textbf{Author(s)}

Søren Højsgaard, <sorenh@math.aau.dk>
set-jevidence

References


See Also

setFinding getFinding retractFinding pFinding

Examples

e = example("grain")

uni <- list(asia = c("yes", "no"), tub = c("yes", "no"),
  smoke = c("yes", "no"), lung = c("yes", "no"),
  bronc = c("yes", "no"), either = c("yes", "no"),
  xray = c("yes", "no"), dysp = c("yes", "no"))

ev <- list(tab("asia", levels=uni, values=c(1,0)),
  tab("dysp", levels=uni, values=c(1,0)),
  tab(c("dysp","bronc"), levels=uni, values=c(.1,.2,.9,.8)))

bn2 <- setjevi(bn, evidence=ev)

## Notice: The evidence is defined on (subsets of) cliques of the junction tree
## and therefore evidence can readily be absorbed:
getgrain(bn, "rip")$cliques  %>% str

## On the other hand, below evidence is not defined cliques of the
## junction tree and therefore evidence can not easily be absorbed.
## Hence this will fail:

## Not run:
ev.fail <- list(tab(c("dysp","smoke"), levels=uni, values=c(.1,.2,.9,.8)))
setjevi(bn, evidence=ev.fail)

## End(Not run)

## Evidence can be removed with

retractjevi(bn2)  ## All evidence removed.
retractjevi(bn2, 0) ## No evidence removed.
retractjevi(bn2, 1:2) ## Evidence items 1 and 2 are removed.

## Setting additional joint evidence to an object where joint
## evidence already is set will cause an error. Hence this will fail:
## Not run:
ev2 <- list(smoke="yes")
setjevi(bn2, evidence=ev2)

## End(Not run)
## update.CPTgrain

### Description

Update a Bayesian network

### Usage

```r
## S3 method for class 'CPTgrain'
update(object, ...)```

### Arguments

- **object**: A Bayesian network of class CPTgrain
- **...**: If CPTlist is a name in the dotted list, then the object will be update with this value (which is assumed to be a list of conditional probabilities). ... here~

### Value

A new Bayesian network. If it is a LIST, use

### Note

There is NO checking that the input matches the settings in the Bayesian network.
Author(s)
Søren Højsgaard, <sorenh@math.aau.dk>

References

Examples

```r
## Network for Bernulli experiment; two nodes: X and thetaX
yn <- c("yes", "no") # Values for X
thX.val <- c(.3, .5, .7) # Values for thetaX
prX.val <- rep(1, length(thX.val)) # Probabilities for thetaX values

thX <- cptable(-thetaX, values=prX.val, levels=thX.val)
X <- cptable(-X|thetaX, values=rbind(thX.val, 1-thX.val), levels=yn)

cptlist <- compileCPT( list(thX, X) )
bn <- compile( grain( cptlist ) )
querygrain( setEvidence(bn, nodes="X", states="yes") )

## To insert a new prior distribution we may do as follows
## (where we can omit the process of recompiling the network)
prX.val2 <- c(.2, .3, .5)
thX2 <- cptable(-thetaX, values=prX.val2, levels=thX.val)
bn2 <- update(bn, CPTlist=compileCPT( list(thX2, X)))
querygrain( setEvidence(bn2, nodes="X", states="yes") )
```
grain-propagate, 20
grain-simulate, 22
grain.CPTspec (grain-main), 17
grain.dModel (grain-main), 17
grain.graphNEL (grain-main), 17
grain.POTspec (grain-main), 17
insertEvi (grain-evi), 11
insertJEvi (set-jevidence), 30
iplot.grain (grain-main), 17
is.grain (grain-main), 17
is.null_ev (evidence-object), 4
junctionTree, 11
load-save-hugin, 23
loadHuginNet (load-save-hugin), 23
logical, 24
mendel, 26
new_ev (evidence-object), 4
new_jev (set-jevidence), 30
nodeNameNames (grain-generics), 16	nodeStates (grain-generics), 16
ortab (logical), 24
ortable, 4
ortable (logical), 24
pEvidence, 9, 12, 27
pEvidence (grain-evi), 11
pFinding, 12, 15, 18, 31
pFinding (finding), 8
plot.grain (grain-main), 17
predict.grain (grain-predict), 19
print.CPTspec (compile-cpt), 2
print.grain_ev (evidence-object), 4
print.grain_jev (set-jevidence), 30
propagate, 11
propagate.grain, 18
propagate.grain (grain-propagate), 20
propagate_ (grain-propagate), 20
propagateLS (grain-propagate), 20
propagateLS_ (grain-propagate), 20
qgrain (querygrain), 26
querygrain, 9, 26
repeatPattern, 28
retractEvi (grain-evi), 11
retractEvi_ (grain-evi), 11
retractEvidence, 9, 12, 27
retractEvidence (grain-evidence), 14
retractFinding, 12, 15, 18, 31
retractFinding (finding), 8
retractJEvi (set-jevidence), 30
rip, 11
saveHuginNet (load-save-hugin), 23
set-jevidence, 30
setdiff_ev (evidence-object), 4
setEv (grain-evi), 11
setEv_ (grain-evi), 11
setEvidence, 9, 12, 18, 27
setEvidence (grain-evidence), 14
setFinding, 12, 15, 18, 31
setFinding (finding), 8
setJEvi (set-jevidence), 30
setJEvi_ (set-jevidence), 30
simulate.grain (grain-simulate), 22
subset.grain_ev (evidence-object), 4
summary.CPTspec (compile-cpt), 2
triangulate, 11
union_ev (evidence-object), 4
universe (grain-generics), 16
update.CPTgrain, 32
varNames.grain_ev (evidence-object), 4
varNames.grainEvidence_ (grain-generics), 16