Bayesian networks in R with the **gRain** package

Søren Højsgaard
Aalborg University, Denmark

gRain version 1.3-5 as of 2020-06-14

Contents

1 Introduction

The **gRain** package implements Bayesian Networks (hereafter often abbreviated BNs). The name **gRain** is an acronym for [gra]phical [i]ndependence [n]etworks. The main reference for **gRain** to cite is Højsgaard (2012), see also

```
citation("gRain")
##
## To cite gRain in publications use:
##
##   Søren Højsgaard (2012). Graphical Independence Networks with the gRain
##   Package for R. Journal of Statistical Software, 46(10), 1-26. URL
##   http://www.jstatsoft.org/v46/i10/.
##
```

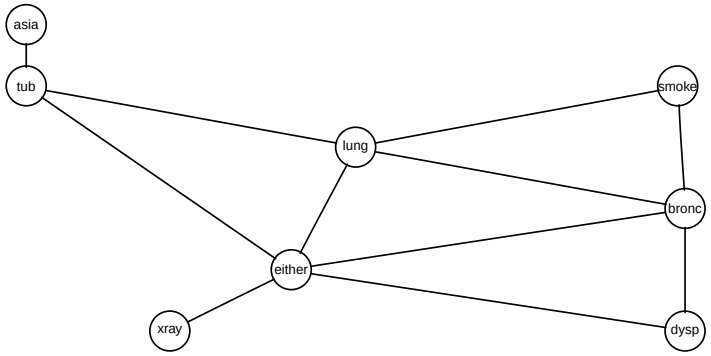


Figure 1: Chest clinic example from Lauritzen and Spiegelhalter (1988).

```
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Graphical Independence Networks with the {gRain} Package for {R}},
##     author = {S{\o}ren H{\o}jsgaard},
##     journal = {Journal of Statistical Software},
##     year = {2012},
##     volume = {46},
##     number = {10},
##     pages = {1--26},
##     url = {http://www.jstatsoft.org/v46/i10/},
##   }
```

Moreover, Højsgaard et al. (2012) gives a broad treatment of graphical models (including Bayesian networks) More information about the package, other graphical modelling packages and development versions is available from

`http://people.math.aau.dk/~sorenh/software/gR`

2 Example: Chest clinic

This section reviews the chest clinic example of Lauritzen and Spiegelhalter (1988) (illustrated in Figure 1) and shows one way of specifying the model in **gRain**. Lauritzen and Spiegelhalter (1988) motivate the chest clinic example with the following narrative:

> "Shortness–of–breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis. The results of a single chest X–ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea."

2.1 Building a network

The description above involves the following binary variables: α = asia, σ = smoker, τ = tuberculosis, λ = lung cancer, β = bronchitis, ϵ = either tuberculosis or lung cancer, δ = dyspnoea

and ξ = xray. Each variable is binary and can take the values "yes" and "no": Note that ϵ is a logical variable which is true (yes) if either τ or λ are true (yes) and false (no) otherwise. The connection between the variables is displayed by the DAG (directed acyclic graph) in Figure 1.

A joint probability density factorising accoring to a DAG with nodes V can be constructed as follows: Each node $v \in V$ has a set $pa(v)$ of parents and each node $v \in V$ has a finite set of states. A joint distribution over the variables V can be given as

$$p(V) = \prod_{v \in V} p(v|pa(v)) \tag{1}$$

where $p(v|pa(v))$ is a function defined on $(v, pa(v))$. This function satisfies that $\sum_{v^*} p(v = v^*|pa(v)) = 1$, i.e. that for each configuration of the parents $pa(v)$, the sum over the levels of v equals one. Hence $p(v|pa(v))$ becomes the conditional distribution of v given $pa(v)$. In practice $p(v|pa(v))$ is specified as a table called a conditional probability table or a CPT for short. Thus, a Bayesian network can be regarded as a complex stochastic model built up by putting together simple components (conditional probability distributions). A joint probability density for all eight variables in Figure 1 can be constructed as

$$p(V) = p(\alpha)p(\sigma)p(\tau|\alpha)p(\lambda|\sigma)p(\beta|\sigma)p(\epsilon|\tau, \lambda)p(\delta|\epsilon, \beta)p(\xi|\epsilon). \tag{2}$$

2.2 Queries to networks

Suppose we are given the evidence (sometimes also called "finding") that a set of variables $E \subset V$ have a specific value e^*. With this evidence, we are often interested in the conditional distribution $p(v|E = e^*)$ for some of the variables $v \in V \setminus E$ or in $p(U|E = e^*)$ for a set $U \subset V \setminus E$. Interest might also be in calculating the probability of a specific event, e.g. the probability of seeing a specific evidence, i.e. $p(E = e^*)$. Other types of evidence (called soft evidence, virtual evidence or likelihood evidence) are discussed in Section 4.

For example that a person has recently visited Asia and suffers from dyspnoea, i.e. α = yes and δ = yes. In the chest clinic example, interest might be in $p(\lambda|e^*)$, $p(\tau|e^*)$ and $p(\beta|e^*)$, or possibly in the joint (conditional) distribution $p(\lambda, \tau, \beta|e^*)$.

3 A one–minute version of gRain

3.1 Specifying a network

A simple way of specifying the model for the chest clinic example is as follows.

1. Specify conditional probability tables (with values as given in Lauritzen and Spiegelhalter (1988)) (there are other ways of specifying conditional probability tables, see the package documentation):

```
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)
```

2. Compile list of conditional probability tables.

```
chest_cpt <- compileCPT(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)
summary(chest_cpt)
##        Length Class  Mode
## asia   2      -none- numeric
## tub    4      -none- numeric
## smoke  2      -none- numeric
## lung   4      -none- numeric
## bronc  4      -none- numeric
## either 8      -none- numeric
## xray   4      -none- numeric
## dysp   8      -none- numeric
```

The components are arrays, but coercion into dataframes sometimes makes it easier to digest the components.

```
chest_cpt$tub
##      asia
## tub    yes   no
##   yes 0.05 0.01
##   no  0.95 0.99
chest_cpt$tub  %>% as.data.frame.table
##   tub asia Freq
## 1 yes  yes 0.05
## 2  no  yes 0.95
## 3 yes   no 0.01
## 4  no   no 0.99
```

Notice: `either` is a logical node

```
chest_cpt$either  %>% as.data.frame.table
##   either lung tub Freq
## 1    yes  yes yes    1
## 2     no  yes yes    0
## 3    yes   no yes    1
## 4     no   no yes    0
## 5    yes  yes  no    1
## 6     no  yes  no    0
## 7    yes   no  no    0
## 8     no   no  no    1
```

3. Create the network:[1]

```
chest_bn <- grain(chest_cpt)
chest_bn
## Independence network: Compiled: TRUE Propagated: FALSE
##   Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray" "dysp"
```

Compile the network (see references for details about this):[2]

[1] SH: Rethink print method

[2] SH: Maybe change so that default is that a network is compiled on creation time.

```
chest_bn <- compile(chest_bn)
```

3.2 Querying a network

1. The network can be queried to give marginal probabilities:[3]

```
querygrain(chest_bn, nodes=c("lung", "bronc"), type="marginal")
## $lung
## lung
##   yes    no
## 0.055 0.945
##
## $bronc
## bronc
##  yes   no
## 0.45 0.55
```

2. Likewise, a joint distribution can be obtained:

```
querygrain(chest_bn, nodes=c("lung", "bronc"), type="joint")
##      bronc
## lung     yes     no
##   yes 0.0315 0.0235
##   no  0.4185 0.5265
```

3. Evidence can be entered in one of these two equivalent forms:

```
chest_bn2  <- setEvidence(chest_bn, evidence=list(asia="yes", dysp="yes"))
chest_bn2  <- setEvidence(chest_bn,
                     nodes=c("asia", "dysp"), states=c("yes", "yes"))
```

4. The probability of observing this evidence under the model is

```
pEvidence(chest_bn2)
## [1] 0.004501375
```

5. The network can be queried again:[4]

```
querygrain(chest_bn2, nodes=c("lung", "bronc"))
## $lung
## lung
##        yes         no
## 0.09952515 0.90047485
##
## $bronc
## bronc
##       yes        no
## 0.8114021 0.1885979
```

[3] `querygrain()` can be abbreviated `qgrain()`.
[4] SH: FIXME; joint is wrong

```
querygrain(chest_bn2, nodes=c("lung", "bronc"), type="joint")
##      bronc
## lung         yes         no
##   yes 0.06298076 0.03654439
##   no  0.74842132 0.15205354
```

Notice a small shortcut: A common usage of a Bayesian network is to enter evidence and then ask for the conditional distribtion of some variables: This can be accomplished in one simple step as follows:[5]

```
querygrain(chest_bn, evidence=list(asia="yes", dysp="yes"),
           nodes=c("lung", "bronc"), type="joint")
##      bronc
## lung         yes         no
##   yes 0.06298076 0.03654439
##   no  0.74842132 0.15205354
```

3.3 Conditioning on evidence with zero probability

Consider setting the evidence

```
chest_bn3 <- setEvidence(chest_bn, evidence=list(either="no", tub="yes"))
```

Under the model, this specific evidence has zero probability: `either` is true if `tub` is true or `lung` is true (or both). Hence the specific evidence is impossible and therefore, all conditional probabilities are (under the model) undefined:

```
pEvidence(chest_bn3)
## [1] 0
querygrain(chest_bn3, nodes=c("lung", "bronc"), type="joint")
##      bronc
## lung  yes  no
##   yes NaN NaN
##   no  NaN NaN
```

Zero probailities (or almost zero probabilities) also arise in a different in a different setting. Consider this example

```
yn <- c("yes","no")
eps <- 1e-100
a    <- cptable(~a,   values=c(1, eps), levels=yn)
b.a  <- cptable(~b+a, values=c(1, eps, eps, 1), levels=yn)
c.b  <- cptable(~c+b, values=c(1, eps, eps, 1), levels=yn)
plist <- compileCPT(list(a, b.a, c.b))
bn   <- grain(plist)
tt   <- querygrain(bn, type="joint")
ftable(tt)
```

[5] SH: FIXME

```
##         c    yes     no
## a   b
## yes yes    1e+00 1e-100
##     no    1e-200 1e-100
## no  yes   1e-200 1e-300
##     no    1e-200 1e-100
querygrain(setEvidence(bn, evidence=list(a="no", c="yes")))
## $b
## b
## yes  no
## 0.5 0.5
```

No problem so far, but if `eps` is made smaller numerical problems arise:[6]

```
eps  <- 1e-200
a    <- cptable(~a,   values=c(1, eps),levels=yn)
b.a  <- cptable(~b+a, values=c(1, eps, eps, 1),levels=yn)
c.b  <- cptable(~c+b, values=c(1, eps, eps, 1),levels=yn)
plist <- compileCPT(list(a, b.a, c.b))
bn   <- grain(plist)
tt   <- querygrain(bn, type="joint")
ftable(tt)
##         c    yes     no
## a   b
## yes yes    1e+00 1e-200
##     no     0e+00 1e-200
## no  yes    0e+00  0e+00
##     no     0e+00 1e-200
querygrain(setEvidence(bn, evidence=list(a="no", c="yes")))
## $b
## b
## yes  no
## NaN NaN
```

3.4 Brute force computations and why they fail

The **gRain** package makes computations as those outlined above in a very efficient way; please see the references. However, it is in this small example also possible to make the computations directly: We can construct the joint distribution (an array with $2^8 = 256$ entries) directly as:

```
joint <- ar_prod_list(chest_cpt)
dim(joint)
## [1] 2 2 2 2 2 2 2 2
joint  %>% as.data.frame.table %>% head
##   xray asia smoke lung tub dysp bronc either        Freq
## 1  yes  yes   yes  yes yes  yes   yes    yes 1.32300e-05
## 2   no  yes   yes  yes yes  yes   yes    yes 2.70000e-07
## 3  yes   no   yes  yes yes  yes   yes    yes 2.61954e-04
## 4   no   no   yes  yes yes  yes   yes    yes 5.34600e-06
## 5  yes  yes    no  yes yes  yes   yes    yes 6.61500e-07
## 6   no  yes    no  yes yes  yes   yes    yes 1.35000e-08
```

[6] Her vil det netop vaere smart at laegge tabeller ind separat!!

This will clearly fail even moderate size problems: For example, a model with 80 nodes each with 10 levels will give a joint state space with 10^{80} states; that is about the number of atoms in the universe. Similarly, 265 binary variables will result in a joint state space of about the same size. Yet, **gRain** has been used succesfully in models with tens of thousand variables. The "trick" in **gRain** is to make all computations without ever forming the joint distribution.

However, we can do all the computations by brute force methods as we will illustrate here:

Marginal distributions are

```
ar_marg(joint, "lung")
## lung
##   yes    no
## 0.055 0.945
ar_marg(joint, "bronc")
## bronc
##  yes   no
## 0.45 0.55
```

Conditioning on evidence can be done in different ways: The conditional density is a 6–way slice of the original 8–way joint distribution:

```
ev <- list(asia="yes", dysp="yes")
cond1 <- ar_slice(joint, slice=ev)
cond1 <- cond1 / sum(cond1)
dim(cond1)
## [1] 2 2 2 2 2 2
ar_marg(cond1, "lung")
## lung
##        yes         no
## 0.09952515 0.90047485
ar_marg(cond1, "bronc")
## bronc
##       yes        no
## 0.8114021 0.1885979
```

Alternatively, multiply all entries not consistent by zero and all other entries by one and then marginalize:

```
cond2 <- ar_slice_mult(joint, slice=ev)
cond2 <- cond2 / sum(cond2)
dim(cond2)
## [1] 2 2 2 2 2 2 2 2
ar_marg(cond2, "lung")
## lung
##        yes         no
## 0.09952515 0.90047485
ar_marg(cond2, "bronc")
## bronc
##       yes        no
## 0.8114021 0.1885979
```

4 Hard and virtual (likelihood) evidence

Below we describe how to work with virtual evidence (also known as likelihood evidence) in **gRain**. This is done via the function `setEvidence()`.

The clique potential representation in a Bayesian network gives

$$p(x) \propto \psi(x) = \prod_C \psi_C(x_C)$$

where we recall that the whole idea in computations with Bayesian networks is to avoid calculation the product on the right hand side. Instead computations are based on propagation (multiplying, dividing and summing clique potentials ψ_C in an appropriate order, and such an appropriate order comes from a junction tree). The normalizing constant, say $c = \sum_x \psi(x)$, comes out of propagation as a "by product".

Suppose a set of nodes E are known to have a specific value, i.e. $x_E = x_E^*$. This is called hard evidence. The probability of the event $x_E = x_E^*$ is

$$p(x_E = x_E^*) = E_p\{I(x_E = x_E^*)\} = \sum_x I(x_E = x_E^*)p(x) = \frac{1}{c}\sum_x I(x_E = x_E^*)\psi(x)$$

The computations are based on modifying the clique potentials ψ_C by giving value zero to states in ψ_C which are not consistent with $x_E = x_E^*$. This can be achieved with an indicator function, say $L_C(x_C)$ such that we obtain a set of new potentials $\tilde{\psi}_C = L_C(x_C)\psi_C(x_C)$. Propagation with these new potentials gives, as a by product, $\tilde{c} = \sum \tilde{\psi}(x)$ where $\tilde{\psi}(x) = \prod_C \tilde{\psi}_C(x_C)$. Consequently, we have $p(x_E = x_E^*) = \tilde{c}/c$.

In a more general setting we may have non–negative weights $L(x)$ for each value of x. We may calculate

$$E_p\{L(X)\} = \sum_x L(x)p(x)$$

If $L(X)$ factorizes as $L(X) = L_C(X_C)$ then the computations are carried out as outlined above, i.e. by the message passing scheme.

4.1 An excerpt of the chest clinic network

Consider the following excerpt of the chest clinic network which is described in the paper mentioned above.

```
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)

(plist1 <- compileCPT(list(a, t.a)))
## cpt_spec with probabilities:
##  P( asia )
##  P( tub | asia )
plist1[[1]]
## asia
##  yes   no
## 0.01 0.99
plist1[[2]]
```

```
##      asia
## tub    yes   no
##   yes 0.05 0.01
##   no  0.95 0.99
(chest1 <- grain(plist1))
## Independence network: Compiled: TRUE Propagated: FALSE
##   Nodes: chr [1:2] "asia" "tub"
querygrain(chest1)
## $asia
## asia
##  yes   no
## 0.01 0.99
##
## $tub
## tub
##    yes     no
## 0.0104 0.9896
```

4.2 Specifying hard evidence

Suppose we want to make a diagnosis about tuberculosis given the evidence that a person has recently been to Asia. The functions `setFinding()` (which has been in **gRain** for years) and `setEvidence()` (which is a recent addition to **gRain**) can both be used for this purpose. The following forms are equivalent (`setFinding()` is kept in **gRain** for backward compatibility):

```
setEvidence(chest1, evidence=list(asia="yes"))
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:2] "asia" "tub"
##   Evidence:
##   nodes is.hard.evidence hard.state
## 1  asia             TRUE        yes
##   pEvidence: 0.010000
setEvidence(chest1, nodes="asia", states="yes")
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:2] "asia" "tub"
##   Evidence:
##   nodes is.hard.evidence hard.state
## 1  asia             TRUE        yes
##   pEvidence: 0.010000
## setFinding(chest1, nodes="asia", states="yes")
```

```
querygrain(setEvidence(chest1, evidence=list(asia="yes")))
## $tub
## tub
##  yes   no
## 0.05 0.95
```

4.3 What is virtual evidence (also called likelihood evidence) ?

Suppose we do not know with certainty whether a patient has recently been to Asia (perhaps the patient is too ill to tell). However the patient (if he/she is Caucasian) may be unusually tanned

and this lends support to the hypothesis of a recent visit to Asia.

To accommodate we create an extended network with an extra node for which we enter evidence. However, it is NOT necessary to do so in practice, because we may equivalently enter the virtual evidence in the original network.

We can then introduce a new variable `guess.asia` with `asia` as its only parent.[7]

```
g.a <- parray(c("guess.asia", "asia"), levels=list(yn, yn),
              values=c(.8,.2, .1,.9))
```

This reflects the assumption that for patients who have recently been to Asia we would guess so in 80% of the cases, whereas for patients who have not recently been to A we would (erroneously) guess that they have recently been to Asia in 10% of the cases.

```
(plist2 <- compileCPT(list(a, t.a, g.a )))
## cpt_spec with probabilities:
##  P( asia )
##  P( tub | asia )
##  P( guess.asia | asia )
(chest2 <- grain(plist2))
## Independence network: Compiled: TRUE Propagated: FALSE
##   Nodes: chr [1:3] "asia" "tub" "guess.asia"
querygrain( chest2 )
## $asia
## asia
##  yes   no
## 0.01 0.99
##
## $tub
## tub
##    yes     no
## 0.0104 0.9896
##
## $guess.asia
## guess.asia
##   yes    no
## 0.107 0.893
```

Now specify the guess or judgment, that the person has recently been to Asia:

```
querygrain(setEvidence(chest2, evidence=list(guess.asia="yes")))
## $asia
## asia
##        yes         no
## 0.07476636 0.92523364
##
## $tub
## tub
##        yes         no
## 0.01299065 0.98700935
```

[7] FIXME: Hvorfor vil parray ikke gaa vaek...

4.4 Specifying virtual evidence

The same guess or judgment can be specified as virtual evidence (also called likelihood evidence) for the original network:

```
querygrain(setEvidence(chest1, evidence=list(asia=c(.8, .1))))
## $tub
## tub
##        yes         no
## 0.01299065 0.98700935
```

This also means that specifying that specifying `asia='yes'` can be done as

```
querygrain(setEvidence(chest1, evidence=list(asia=c(1, 0))))
## $tub
## tub
##  yes   no
## 0.05 0.95
```

4.5 A mixture of a discrete and a continuous variable

gRain only handles discrete variables with a finite state space, but using likelihood evidence it is possible to work with networks with both discrete and continuous variables (or other types of variables). This is possible only when he networks have a specific structure. This is possible when no discrete variable has non–discrete parents.[8]

Take a simple example: x is a discrete variable with levels 1 and 2; $y_1|x = k \sim N(\mu_k, \sigma_k^2)$ and $y_2|x = k \sim Poi(\lambda_k)$ where $k = 1, 2$. The joint distribution is

$$p(x, y_1, y_2) = p(x)p(y_1|x)p(y_2|x)$$

Suppose the interest is in the distribution of x given $y_1 = y_1^*$ and $y_2 = y_2^*$. We then have

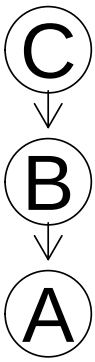
$$p(x|y_1^*, y_2^*) \propto p(x)p(y_1^*|x)p(y_2^*|x) = p(x)L_1(x)L_2(x)$$

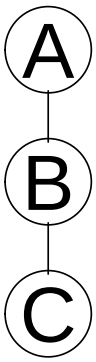
5 Building networks from data

The following two graphs specify the same model:

```
dG  <- dag(~A:B + B:C)
uG  <- ug(~A:B + B:C)
par(mfrow=c(1,2)); plot( dG ); plot( uG )
```

[8] SH: Expand this.





Suppose data is

```
dat <- ar_new(c("A", "B", "C"), levels=c(2, 2, 2), values=c(0, 0, 2, 3, 1, 2, 1, 4))
class(dat)
## [1] "array"
```

A network can be built from data using:

```
gr.dG <- compile( grain( dG, data=dat ) )
gr.uG <- compile( grain( uG, data=dat ) )
```

However, when there are zeros in the table, care must be taken.

5.1 Extracting information from tables

In the process of creating networks, conditional probability tables are extracted when the graph is a dag and clique potentials are extracted when the graph is a chordal (i.e. triangulated) undirected graph. This takes place as follows (internally):

```
extractCPT(dat, dG)
## $A
##     B
## A           B1  B2
##   A1 0.3333333 0.3
##   A2 0.6666667 0.7
##
## $B
##     C
## B    C1    C2
##   B1  0 0.375
##   B2  1 0.625
##
## $C
## C
##        C1        C2
## 0.3846154 0.6153846
##
## attr(,"graph")
## A graphNEL graph with directed edges
```

```
## Number of Nodes = 3
## Number of Edges = 2
## attr(,"class")
## [1] "cpt_rep"
c(extractPOT(dat, uG ))
## [[1]]
##     A
## B            A1        A2
##   B1 0.07692308 0.1538462
##   B2 0.23076923 0.5384615
##
## [[2]]
##     B
## C    B1  B2
##   C1  0 0.5
##   C2  1 0.5
```

The conditional probability table $P(A|B)$ contains `NaN`s because

$$P(A|B = B1) = \frac{n(A, B = B1)}{\sum_A n(A, B = B1)} = \frac{0}{0} = \text{NaN}$$

For this reason the network `gr.dG` above will fail to compile whereas `gr.uG` will work, but it may not give the expected results.

5.2 Using smooth

To illustrate what goes on, we can extract the distributions from data as follows:[9]

```
p.A.g.B <- tableDiv(dat, tableMargin(dat, "B"))
p.B     <- tableMargin(dat, "B") / sum(dat)
p.AB    <- tableMult( p.A.g.B, p.B)
```

However, the result is slightly misleading because `tableDiv` sets $0/0 = 0$.

In `grain` there is a `smooth` argument that will add a small number to the cell entries before extracting tables, i.e.

$$P(A|B = B1) = \frac{n(A, B = B1) + \epsilon}{\sum_A (n(A, B = B1) + \epsilon)} = \frac{\epsilon}{2\epsilon} = 0.5$$

and

$$P(B) = \frac{\sum_A (n(A, B) + \epsilon)}{\sum_{AB} (n(A, B) + \epsilon)}$$

We can mimic this as follows:

```
e <- 1e-2
(dat.e <- dat + e)
## , , C = C1
##
##     B
```

[9] SH: FIXME Use new functions from gRbase.

```
## A      B1   B2
##   A1 0.01 2.01
##   A2 0.01 3.01
##
## , , C = C2
##
##     B
## A      B1   B2
##   A1 1.01 1.01
##   A2 2.01 4.01
```

```
pe.A.g.B <- tableDiv(dat.e, tableMargin(dat, "B"))
pe.B <- tableMargin(dat.e, "B")/sum(dat.e)
pe.AB  <- tableMult( pe.A.g.B, pe.B )
```

However this resulting joint distribution is different from what is obtained from the adjusted table itself

```
dat.e / sum(dat.e)
## , , C = C1
##
##     B
## A             B1        B2
##   A1 0.000764526 0.1536697
##   A2 0.000764526 0.2301223
##
## , , C = C2
##
##     B
## A            B1         B2
##   A1 0.07721713 0.07721713
##   A2 0.15366972 0.30657492
```

This difference appears in the **gRain** networks.

5.3 Extracting tables

One can do

```
gr.dG <- compile(grain(dG, data=dat, smooth=e))
```

which (internally) corresponds to

```
extractCPT(dat, dG, smooth=e)
## $A
##     B
## A           B1        B2
##   A1 0.3344371 0.3003992
##   A2 0.6655629 0.6996008
##
## $B
```

```
##     C
## B             C1        C2
##   B1 0.001992032 0.3753117
##   B2 0.998007968 0.6246883
##
## $C
## C
##        C1        C2
## 0.3847926 0.6152074
##
## attr(,"graph")
## A graphNEL graph with directed edges
## Number of Nodes = 3
## Number of Edges = 2
## attr(,"class")
## [1] "cpt_rep"
```

We get

```
querygrain(gr.dG)
## $A
## A
##        A1        A2
## 0.3082845 0.6917155
##
## $B
## B
##        B1        B2
## 0.2316611 0.7683389
##
## $C
## C
##        C1        C2
## 0.3847926 0.6152074
querygrain(gr.uG)
## $B
## B
##        B1        B2
## 0.2307692 0.7692308
##
## $A
## A
##        A1        A2
## 0.3076923 0.6923077
##
## $C
## C
##        C1        C2
## 0.3846154 0.6153846
```

However, if we condition on `B=B1` we get:

```
querygrain(setFinding(gr.dG, nodes="B", states="B1"))
## $A
## A
##        A1        A2
## 0.3344371 0.6655629
##
## $C
## C
##          C1          C2
## 0.003308796 0.996691204
querygrain(setFinding(gr.uG, nodes="B", states="B1"))
## $A
## A
##        A1        A2
## 0.3333333 0.6666667
##
## $C
## C
## C1 C2
##  0  1
```

so the "problem" with zero entries shows up in a different place. However, the answer is not necessarily wrong; the answer simply states that $P(A|B = B1)$ is undefined. To "remedy" we can use the `smooth` argument:

```
gr.uG <- compile(grain(uG, data=dat, smooth=e))
```

which (internally) corresponds to

```
c(extractPOT(dat, uG, smooth=e))
## [[1]]
##    A
## B            A1        A2
##   B1 0.07745399 0.1541411
##   B2 0.23082822 0.5375767
##
## [[2]]
##    B
## C             B1  B2
##   C1 0.003311258 0.5
##   C2 0.996688742 0.5
```

Notice that the results are not exactly identical:

```
querygrain(gr.uG)
## $B
## B
##        B1        B2
## 0.2315951 0.7684049
##
## $A
## A
```

```
##        A1        A2
## 0.3082822 0.6917178
##
## $C
## C
##        C1        C2
## 0.3849693 0.6150307
querygrain(gr.dG)
## $A
## A
##        A1        A2
## 0.3082845 0.6917155
##
## $B
## B
##        B1        B2
## 0.2316611 0.7683389
##
## $C
## C
##        C1        C2
## 0.3847926 0.6152074
```

```
querygrain(setFinding(gr.uG, nodes="B", states="B1"))
## $A
## A
##        A1        A2
## 0.3344371 0.6655629
##
## $C
## C
##          C1          C2
## 0.003311258 0.996688742
querygrain(setFinding(gr.dG, nodes="B", states="B1"))
## $A
## A
##        A1        A2
## 0.3344371 0.6655629
##
## $C
## C
##          C1          C2
## 0.003308796 0.996691204
```

References

Søren Højsgaard. Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10):1–26, 2012. URL `http://www.jstatsoft.org/v46/i10/`.

Søren Højsgaard, David Edwards, and Steffen L. Lauritzen. *Graphical Models with R*. Springer, 2012. ISBN 978-1-4614-2299-0.

Steffen L. Lauritzen and David Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Roy. Stat. Soc. Ser. B*, 50(2):157–224, 1988.