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

Søren Højsgaard
Aalborg University, Denmark

gRain version 1.3.13 as of 2023-03-09

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** is Højsgaard (2012), see also 'citation("gRain")'.

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`

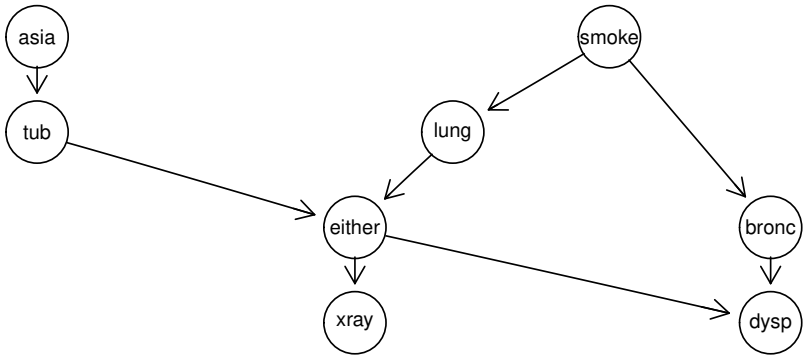


Figure 1: Chest clinic example from Lauritzen and Spiegelhalter (1988).

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 factorizing according 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 6.

For example that a person has recently visited Asia and suffers from dyspnoea, i.e. $\alpha =$ yes and $\delta =$ 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    <- 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)
```

2. Compile list of conditional probability tables.

```
chest_cpt <- compileCPT(a, t.a, s, l.s, b.s, e.lt, x.e, d.be)
chest_cpt
##  P( asia )
##  P( tub | asia )
##  P( smoke )
##  P( lung | smoke )
##  P( bronc | smoke )
##  P( either | lung tub )
##  P( xray | either )
##  P( dysp | bronc either )
```

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:

```
chest_bn <- grain(chest_cpt)
chest_bn
## Independence network: Compiled: TRUE Propagated: FALSE Evidence: FALSE
```

Default is that the network is compiled at creation time, but if one chooses not to do so, compilation can be done with:

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

3.2 Setting evidence and querying a network

1. A network can be queried to give marginal probabilities for each of a set of nodes (the default) or the joint probability for a set of nodes.[1] Notice that `querygrain()` can be abbreviated `qgrain()`.

```
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
```

[1] A third type of output exists, see package documentation for details.

2. Likewise, a joint distribution can be obtained (represented as a multi dimensional array):

```
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 two different ways:[2]

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

## Also: modify object with
## evidence(chest_bn) <- list(asia="yes", dysp="yes")
```

4. The evidence is a list and can conveniently be displayed as a dataframe:

```
getEvidence(chest_ev) |> as.data.frame()
##   nodes is_hard hard_state evi_weight
## 1  asia    TRUE        yes       1, 0
## 2  dysp    TRUE        yes       1, 0
```

5. The network can be queried again:

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

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

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

7. The probability of an evidence can be found with only propagation towards the root of a junction tree. This saves about half the computational effort of propagation. However, notice that the network is not changed by this operation, so if the network is subsequently queried, there is no gain in computing time.

[2] Alternative forms exist; see package documentation for details.

```
pEvidence(chest_bn, evidence=list(asia="yes", dysp="yes"))
## [1] 0.004501375
```

4 Hints and shortcuts

1. An alternative way of specifying a network is by first defining CPTs and then entering values afterwards. Programmatically, this can be done as:

```
yn <- c("yes","no")
flist <- c(
    ~asia, ~tub|asia, ~smoke, ~lung|smoke, ~bronc|smoke, ~either|lung:tub,
    ~xray|either, ~dysp|bronc:either
)
## or
flist <- list("asia", c("tub", "asia"), "smoke", c("lung", "smoke"),
              c("bronc", "smoke"), c("either", "tub", "lung"),
              c("xray", "either"), c("dysp", "bronc", "either"))

chest_cpt2 <- lapply(flist, function(f){
    cpt(f, levels=yn)
})

bn2 <- compileCPT(chest_cpt2) |> grain()

lst2 <- list(asia=c(1, 99),
             tub=c(5, 95, 1, 99),
             smoke=c(5, 5),
             lung=c(1, 9, 1, 99),
             bronc=c(6, 4, 3, 7),
             either=c(1, 0, 1, 0, 1, 0, 0, 1),
             xray=c(98, 2, 5, 95),
             dysp=c(9, 1, 7, 3, 8, 2, 1, 9))

bn2 <- replaceCPT(bn2, lst2)
```

2. Consider querying a network where focus is on marginal distributions (the default). If all variables have the same levels (as the case is here), the output can be coerced to a dataframe:

```
querygrain(chest_bn, nodes=c("lung", "bronc"), simplify = TRUE)
##         yes    no
## lung  0.055 0.945
## bronc 0.450 0.550
```

In the more general case the output can be coerced to a list of dataframes

```
querygrain(chest_bn, nodes=c("lung", "bronc"), result="data.frame")
## $lung
##   lung  Freq
## 1  yes 0.055
## 2   no 0.945
```

```
##
## $bronc
##   bronc Freq
## 1   yes 0.45
## 2    no 0.55
```

3. A typical use of Bayesian network involves setting evidence and then querying the network afterwards. This can all be done in one call of `querygrain` (notice that this does not alter the network object):

```
querygrain(chest_bn,
           evidence=list(asia="yes", dysp="yes"),
           nodes=c("lung", "bronc"), simplify = TRUE)
##              yes        no
## lung  0.09952515 0.9004749
## bronc 0.81140207 0.1885979
```

4. Evidence can be also be given as a vector of weights.

```
querygrain(chest_bn,
           evidence=list(asia=c(1,0), dysp=c(1,0)),
           nodes=c("lung", "bronc"), simplify = TRUE)
##              yes        no
## lung  0.09952515 0.9004749
## bronc 0.81140207 0.1885979
```

   The weights must be non-negative but need not sum to one. This is important in connection with soft evidence (also called likelihood evidence), see Section 6. Above, the weights could also have been set as `c(.1, 0)`. The important part is that the zero excludes certain states as being impossible.

5. Nodes on which evidence is given are not reported unless `exclude=FALSE`

```
querygrain(chest_bn,
           evidence=list(asia=c(1, 0), dysp=c(1, 0)),
           nodes=c("lung", "bronc", "asia", "dysp"),
           exclude=FALSE, simplify = TRUE)
##              yes        no
## asia  1.00000000 0.0000000
## lung  0.09952515 0.9004749
## bronc 0.81140207 0.1885979
## dysp  1.00000000 0.0000000
```

6. If `nodes` are not specified, queries for all nodes without evidence are reported.

```
querygrain(chest_bn,
           evidence=list(asia="yes", dysp="yes"),
           simplify = TRUE)
##               yes        no
## tub    0.08775096 0.9122490
## lung   0.09952515 0.9004749
## either 0.18229985 0.8177001
```

```
## bronc  0.81140207 0.1885979
## smoke  0.62591986 0.3740801
## xray   0.21953886 0.7804611
```

If `nodes` are not specified and `exclude=FALSE`, then queries for all nodes are reported.

```
querygrain(chest_bn,
           evidence=list(asia="yes", dysp="yes"),
           exclude = FALSE, simplify = TRUE)
##               yes        no
## asia   1.00000000 0.0000000
## tub    0.08775096 0.9122490
## lung   0.09952515 0.9004749
## either 0.18229985 0.8177001
## bronc  0.81140207 0.1885979
## smoke  0.62591986 0.3740801
## dysp   1.00000000 0.0000000
## xray   0.21953886 0.7804611
```

5 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
```

6 Hard and virtual (likelihood) evidence

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

The clique potential representation in a Bayesian network gives

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

Here we recall that the whole idea in computations with Bayesian networks is to avoid calculation the product on the right hand side above. 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 $\tilde{\psi}_C(x) = L_C(x_C)\psi_C(x_C)$. 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 $L(X) = \prod_C L_C(X_C)$ then the computations are carried out as outlined above, i.e. by the message passing scheme.

6.1 An excerpt of the chest clinic network

Consider the following excerpt of the chest clinic network.

```
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)

plist1 <- compileCPT(list(a, t.a))
chest1 <- grain(plist1)
querygrain(chest1, simplify = TRUE)
##         yes     no
## asia 0.0100 0.9900
## tub  0.0104 0.9896
```

6.2 Specifying hard evidence

Suppose we want to make a diagnosis about tuberculosis given the evidence that a person has recently been to Asia. The function `setEvidence()` can be used for this purpose. The following forms are equivalent

```
setEvidence(chest1, evidence=list(asia="yes"))
## Independence network: Compiled: TRUE Propagated: TRUE Evidence: TRUE
```

We call such evidence hard evidence because the state of the variables are known with certainty.

6.3 Virtual evidence (also called soft or 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 this setting 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.

```
g.a <- cpt(~guess_asia+asia, levels=yn,
              values=c(.8, .2, .1, .9))
```

This reflects the assumption that for patients who have recently been to Asia we would (correctly) 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 ))
chest2 <- grain(plist2)
querygrain(chest2, simplify = TRUE)
##                yes     no
## asia       0.0100 0.9900
## tub        0.0104 0.9896
## guess_asia 0.1070 0.8930
```

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

```
querygrain(chest2, evidence=list(guess_asia="yes"),
           simplify=TRUE, exclude = FALSE)
##                    yes        no
## asia       0.07476636 0.9252336
## tub        0.01299065 0.9870093
## guess_asia 1.00000000 0.0000000
```

6.4 Specifying virtual evidence

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

```
chest1_ve <- chest1 |> setEvidence(evidence=list(asia=c(.8, .1)))
chest1_ve |> querygrain(simplify = TRUE)
##            yes        no
## tub 0.01299065 0.9870093
getEvidence(chest1_ve, short=FALSE)
## $nodes
## [1] "asia"
##
## $is_hard
## [1] FALSE
```

```
##
## $hard_state
## [1] NA
##
## $evi_weight
## $evi_weight[[1]]
## asia
## yes  no
## 0.8 0.1
##
##
## attr(,"class")
## [1] "grain_evidence" "list"
```

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

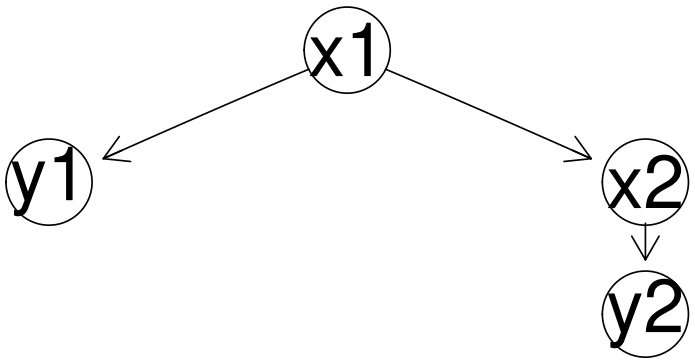
```
querygrain(chest1, evidence=list(asia=c(1, 0)), simplify=T)
##      yes   no
## tub 0.05 0.95
```

6.5 Extending networks to include other types of variables

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.

Take a simple example: Form a Bayesian network for variables $x = (x_1, x_2)$. Conceptually augment this network with additional variables $y = (y_1, y_2)$ where $y_1|x_1 = k \sim N(\mu_k, v)$ and $y_2|x_2 = k \sim Poi(\lambda_k)$ for $k = 1, 2$. Also we make the assumption that y_1 and y_2 are independent given $x = (x_1, x_2)$. This gives the DAG below:

```
plot(dag(~y1:x1 + x2:x1 + y2:x2))
```



A Bayesian network for x can be constructed as:

```
u <- list(x1=yn, x2=yn)
x1 <- cpt(~x1, values=c(1, 3), levels=yn)
x2 <- cpt(~x2|x1, values=c(1, 3, 3, 1), levels=yn)
bn <- grain(compileCPT(x1, x2))
querygrain(bn, simplify=TRUE)
##      yes    no
## x1 0.250 0.750
## x2 0.625 0.375
```

The augmentation of $y|x$ can go along these lines: The parameters describing $y|x$ are set to be:

```
v <- 2
mu <- c(mu1=2, mu2=5)
lambda <- c(lambda1=0, lambda2=7)
```

Suppose we observe $y_1 = y_1^*$. Then

$$p(x|y_1 = y_1^*) \propto p(x_1)p(x_2|x_1)p(y_1 = y_1^*|x_1) = p(x_1)p(x_2|x_1)L_1(x_1)$$

where $L_1(x_1)$ denotes the likelihood. In a Bayesian network setting this corresponds to changing $p(x_1)$ as

$$p(x_1) \leftarrow p(x_1)L_1(x_1)$$

and then carry on with propagation. This can be achieved in different ways. One is by setting the likelihood as evidence:

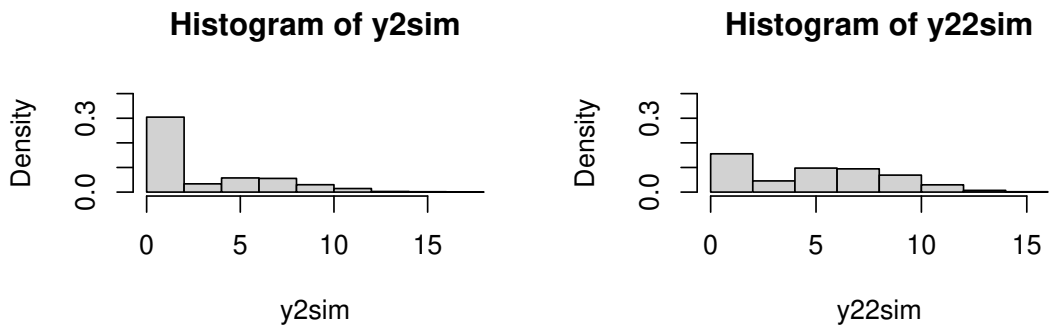
```
y1 <- 1 # Observed value for y1
lik1 <- dnorm(y1, mean=mu, sd=sqrt(v))
querygrain(bn, exclude = FALSE,
           evidence=list(x1=lik1), simplify = TRUE)
##          yes         no
## x1 0.9340965 0.06590353
## x2 0.2829518 0.71704823
```

An alternative is to explicitly modify the CPT which specifies $p(x_1)$:

```
x1_upd <- getgrain(bn, "cptlist")$x1 * lik1
bn2 <- replaceCPT(bn, list(x1=x1_upd))
querygrain(bn2)
## $x1
## x1
##        yes         no
## 0.93409647 0.06590353
##
## $x2
## x2
##       yes        no
## 0.2829518 0.7170482
```

A final remark: The conditional distribution of y_1 is normal, but the unconditional distribution is a mixture of normals. Likewise, the conditional distribution of y_2 is poisson, but the unconditional distribution is a mixture of poissons. Evidence on, say y_1 changes the belief in x_1 and x_2 and this in turn changes the distribution of y_2 (evidence changes the mixture weights.)

```r
set.seed(2022)
nsim <- 1000
xsim1 <- simulate(bn, nsim)
head(xsim1)
##   x1  x2
## 1 no yes
## 2 no yes
## 3 no  no
## 4 no  no
## 5 no yes
## 6 no yes
xsim2 <- simulate(bn2, nsim)
head(xsim2)
##    x1  x2
## 1 yes  no
## 2 yes  no
## 3 yes yes
## 4 yes  no
## 5 yes  no
## 6 yes  no
par(mfrow=c(1,2))
y2sim <- rpois(n=nsim, lambda=lambda[xsim1$x2])
y22sim <- rpois(n=nsim, lambda=lambda[xsim2$x2])
y2sim |> hist(prob=T, ylim=c(0, .4), breaks=10)
y22sim |> hist(prob=T, ylim=c(0, .4), breaks=10)
```



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

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

A 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 )
```

Suppose data is

```
dat <- tabNew(c("A", "B", "C"), levels=c("lev1", "lev2"), #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.

A.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) |> c() ## FIXME: Printing problem
## $A
##       B
## A           lev1 lev2
##   lev1 0.3333333  0.3
##   lev2 0.6666667  0.7
##
## $B
##       C
## B      lev1  lev2
##   lev1    0 0.375
##   lev2    1 0.625
##
```

```
## $C
## C
##      lev1      lev2
## 0.3846154 0.6153846
extractPOT(dat, uG) |> c() ## FIXME: Printing problem
## [[1]]
##       A
## B            lev1      lev2
##   lev1 0.07692308 0.1538462
##   lev2 0.23076923 0.5384615
##
## [[2]]
##       B
## C      lev1 lev2
##   lev1    0  0.5
##   lev2    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.

A.2 Using smooth

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

```
p.A_B <- tabDiv(dat, tabMarg(dat, "B")) ## p(A|B)
p.B   <- tabMarg(dat, "B") / sum(dat)   ## p(B)
p.AB2 <- tabMult(p.A_B, p.B)            ## P(AB)
```

However, the result is slightly misleading because `tabDiv` 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 = lev1
##
##       B
## A      lev1 lev2
##   lev1 0.01 2.01
##   lev2 0.01 3.01
```

```
##
## , , C = lev2
##
##       B
## A      lev1 lev2
##   lev1 1.01 1.01
##   lev2 2.01 4.01
```

```
pe.A_B <- tabDiv(dat.e, tabMarg(dat.e, "B"))
pe.B   <- tabMarg(dat.e, "B") / sum(dat.e)
pe.AB  <- tabMult(pe.A_B, pe.B)
```

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

```
dat2.e <- dat.e / sum(dat.e)
```

```
(dat2.e - pe.AB)  |> ftable()
##           C       lev1       lev2
## A    B
## lev1 lev1    0.0000000 -0.0764526
##      lev2    0.0764526  0.0000000
## lev2 lev1    0.0000000 -0.0764526
##      lev2    0.0764526  0.0000000
```

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

A.3 Extracting tables

One can do

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

which (internally) corresponds to

```
extractCPT(dat, dG, smooth=e)  |> c()
## $A
##       B
## A           lev1      lev2
##   lev1 0.3344371 0.3003992
##   lev2 0.6655629 0.6996008
##
## $B
##       C
## B             lev1      lev2
##   lev1 0.001992032 0.3753117
##   lev2 0.998007968 0.6246883
##
## $C
## C
##      lev1      lev2
## 0.3847926 0.6152074
```

We get

```
querygrain(gr.dG, exclude=FALSE, simplify=TRUE)
##        lev1      lev2
## A 0.3082845 0.6917155
## B 0.2316611 0.7683389
## C 0.3847926 0.6152074
querygrain(gr.uG, exclude=FALSE, simplify=TRUE)
##        lev1      lev2
## B 0.2307692 0.7692308
## A 0.3076923 0.6923077
## C 0.3846154 0.6153846
```

However, if we condition on `B=lev1` we get:

```
querygrain(gr.dG, evidence=list(B="lev1"), exclude=FALSE, simplify=TRUE)
##          lev1      lev2
## A 0.334437086 0.6655629
## B 1.000000000 0.0000000
## C 0.003308796 0.9966912
querygrain(gr.uG, evidence=list(B="lev1"), exclude=FALSE, simplify=TRUE)
##        lev1      lev2
## B 1.0000000 0.0000000
## A 0.3333333 0.6666667
## C 0.0000000 1.0000000
```

so the "problem" with zero entries shows up in a different place. However, the answer is not necessarily wrong; the answer simply states that $P(A|B = lev1)$ is undefined. To "remedy" we can use the `smooth` argument:

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

which (internally) corresponds to

```
extractPOT(dat, uG, smooth=e) |> c()
## [[1]]
##       A
## B            lev1      lev2
##   lev1 0.07745399 0.1541411
##   lev2 0.23082822 0.5375767
##
## [[2]]
##       B
## C             lev1 lev2
##   lev1 0.003311258  0.5
##   lev2 0.996688742  0.5
```

Notice that the results are not exactly identical:

```
querygrain(gr.dG, exclude=FALSE, simplify=TRUE)
##        lev1      lev2
## A 0.3082845 0.6917155
## B 0.2316611 0.7683389
## C 0.3847926 0.6152074
```

```
querygrain(gr.uG, exclude=FALSE, simplify=TRUE)
##        lev1      lev2
## B 0.2315951 0.7684049
## A 0.3082822 0.6917178
## C 0.3849693 0.6150307
```

```
querygrain(gr.dG, evidence=list(B="lev1"), exclude=FALSE, simplify=TRUE)
##          lev1      lev2
## A 0.334437086 0.6655629
## B 1.000000000 0.0000000
## C 0.003308796 0.9966912
querygrain(gr.uG, evidence=list(B="lev1"), exclude=FALSE, simplify=TRUE)
##          lev1      lev2
## B 1.000000000 0.0000000
## A 0.334437086 0.6655629
## C 0.003311258 0.9966887
```

B 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 <- tabListMult(chest_cpt)
dim(joint)
## [1] 2 2 2 2 2 2 2 2
joint  |> as.data.frame.table() |> head()
##   xray smoke asia 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 6.61500e-07
## 4   no    no  yes  yes yes  yes   yes    yes 1.35000e-08
## 5  yes   yes   no  yes yes  yes   yes    yes 2.61954e-04
## 6   no   yes   no  yes yes  yes   yes    yes 5.34600e-06
```

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 successfully 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

```
tabMarg(joint, "lung")
## lung
##   yes    no
## 0.055 0.945
tabMarg(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 <- tabSlice(joint, slice=ev)
cond1 <- cond1 / sum(cond1)
dim(cond1)
## [1] 2 2 2 2 2 2
tabMarg(cond1, "lung")
## lung
##        yes         no
## 0.09952515 0.90047485
tabMarg(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 <- tabSliceMult(joint, slice=ev)
cond2 <- cond2 / sum(cond2)
dim(cond2)
## [1] 2 2 2 2 2 2 2 2
tabMarg(cond2, "lung")
## lung
##        yes         no
## 0.09952515 0.90047485
tabMarg(cond2, "bronc")
## bronc
##       yes        no
## 0.8114021 0.1885979
```

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.