Aspects of Bayesian Networks

Presented for the Aalborg R user group

April 2021

Søren Højsgaard©

Department of Mathematical Sciences

Aalborg University, Denmark

April 8, 2021

Printed: April 8, 2021 File: bayesnet-slides.tex

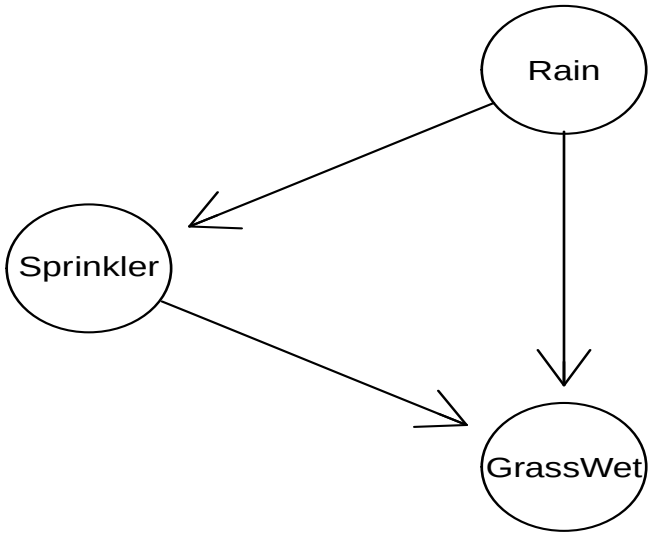
Contents

1 Example: The sprinkler network

1.1 The setting

Two events can cause grass to be wet: Either the sprinkler is on or it is raining. rain has a direct effect on the use of the sprinkler: when it rains, the sprinkler is usually not turned on.

What is the probability that it has been raining given that the grass is wet?



This can be modeled with a Bayesian network. The variables (R)ain, (S)prinkler, (G)rassWet have two possible values: (y)es and (n)o.

In a model building context we start in by defining conditional probabilities specifying the terms

$$P(G|S,R), \quad P(S|R), \quad P(R)$$

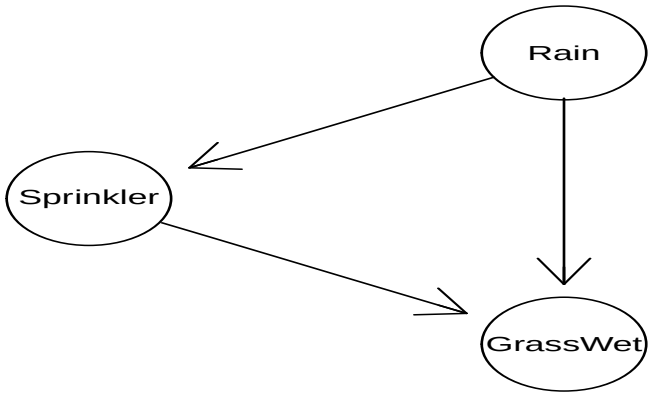
We call these terms conditional probability tables (or CPTs). Then we construct a joint pmf by

$$P(G,S,R) \leftarrow P(G|S,R)P(S|R)P(R)$$

Notice this: Terms on the right hand side above have the form

$$p(v|parent(v))$$

relative to the DAG (directed acyclic graph):



1.2 Conditional probability tables (CPTs)

For compact printing of arrays define utility function

```
> flatten <- function(x){
    ftable(x, row.vars=1)
  }
```

Conditional probability tables (CPTs) in R are arrays. Arrays can be created e.g. with `array()` or as follows:

```
> yn <- c("yes", "no")
> domain <- list(rain = yn, sprinkler = yn, wet_grass = yn)
> ## P(R)
> p.R <- tabNew("rain", levels=domain, values=c(.2, .8))
> p.R
rain
yes  no
0.2 0.8
```

```
> ## P(S|R)
> p.S_R <- tabNew(c("sprinkler", "rain"), levels = domain,
                  values=c(.01, .99, .4, .6))
> p.S_R %>% flatten
          rain  yes   no
sprinkler
yes             0.01 0.40
no              0.99 0.60
> ## P(G|S,R)
> p.G_SR <- tabNew(~wet_grass:sprinkler:rain, levels = domain,
                   values=c(.99, .01, .8, .2, .9, .1, 0, 1))
> p.G_SR  %>% flatten
          sprinkler  yes         no
          rain       yes   no  yes   no
wet_grass
yes                  0.99 0.90 0.80 0.00
no                   0.01 0.10 0.20 1.00
```

1.3 A small digression: Operations on arrays

```
> T1 <- tabNew(~a:b, levels=c(2,2), values=1:4)
> T2 <- tabNew(~b:c, levels=c(2,2), values=5:8)
> T1; T2
    b
a    b1 b2
  a1  1  3
  a2  2  4
    c
b    c1 c2
  b1  5  7
  b2  6  8
```

Think of T_1 as function of variables (a, b) and T_2 as function of (b, c).

The product $T = T_1 T_2$ is a function of (a, b, c) defined as

$$T(a, b, c) \leftarrow T_1(a, b) T_2(b, c)$$

```
> T1 %a*% T2   %>% flatten
   b b1    b2
   c c1 c2 c1 c2
a
a1    5  7 18 24
a2   10 14 24 32
```

1.4 Operations on arrays - overview

```
> ## Multiplication
> T1 %a*% T2
> tabMult(T1, T2)
> ## Division
> T1 %a/% T2
> tabDiv(T1, T2)
> ## Division; 0/0 = 0
> T1 %a/0% T2
> tabDiv0(T1, T2)
> ## Addition
> T1 %a+% T2
> tabAdd(T1, T2)
> ## Subtraction
> T1 %a-% T2
> tabSubt(T1, T2)
> ## Equality
> T1 %a==% T2
> tabEqual(T1, T2)
> ## Marginalization
> T1 %a_% set
> tabMarg(T1, set)
```

Joint pmf:

```
> ## P(G,S,R)
> p.GSR <- p.G_SR %a*% p.S_R %a*% p.R
> p.GSR  %>% flatten
          wet_grass     yes              no
          rain          yes      no      yes      no
sprinkler
yes                   0.00198 0.28800 0.00002 0.03200
no                    0.15840 0.00000 0.03960 0.48000
> sum(p.GSR) # check
[1] 1
```

1.5 Using Bayes' formula

Question: What is the probability that it is raining given that the grass is wet?

Answer: Use Bayes formula:

$$P(R|G=y) = \frac{P(R,G=y)}{P(G=y)} = \frac{\sum_{S=y,n} P(R,S,G=y)}{\sum_{R=y,n;S=y,n} P(R,S,G=y)}$$

This question - and others - can be answered with `tabDist`:

```
> tabDist(p.GSR, marg="rain", cond=list(wet_grass="yes"))
rain
 yes   no
0.36 0.64
```

1.6 Under the hood

In detail we have the following computations:

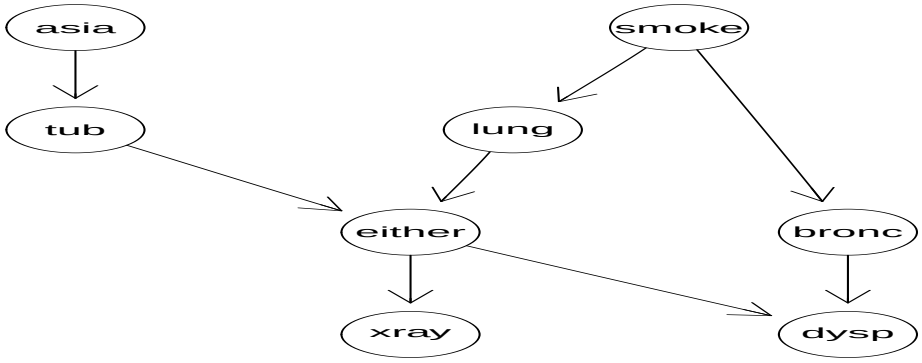
```
> ## 1) Marginalize: P(S, G, R) -> P(R, G) -> P(G)
> p.RG  <- p.GSR %amarg% ~rain + wet_grass; p.RG
      wet_grass
rain    yes   no
  yes 0.16 0.04
  no  0.29 0.51
> p.G   <- p.RG  %amarg% ~wet_grass; p.G
wet_grass
 yes   no
0.45 0.55
> ## 2) Condition -> P(R|G)
> p.R_G <- p.RG %a/% p.G; p.R_G
      wet_grass
rain    yes    no
  yes 0.36 0.072
  no  0.64 0.928
> ## 3) Pick the slice -> P(R|G=yes)
> p.R_G %aslice% list(wet_grass="yes")
 yes   no
0.36 0.64
```

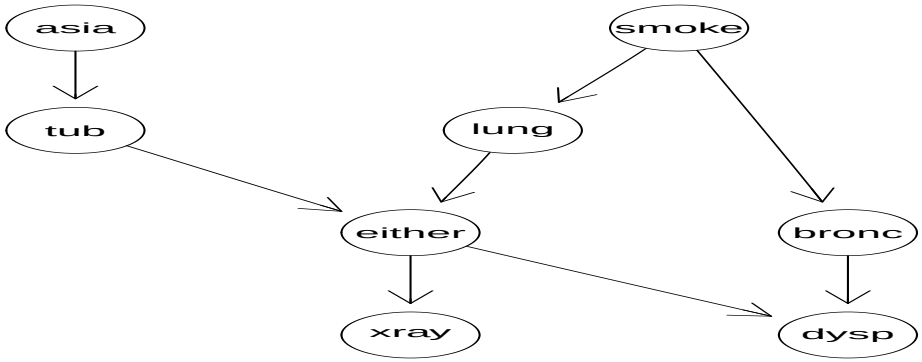
2 Example: The chest clinic narrative

Lauritzen and Spiegehalter (1988) present 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 *bronchitis*, as n*either* does the presence or absence of *dyspnoea*.

The narrative can be pictured as a DAG (Directed Acyclic Graph)





With an informal notation, a joint distribution for all variables

$$\begin{aligned} V &= \{Asia, Tub, Smoke, Lung, Either, Bronc, Xray, Dysp\} \\ &\equiv \{a, t, s, l, e, b, x, d\} \end{aligned}$$

can be obtained as

$$p(V) = \prod_v p(v|pa(v))$$

which here boils down to

$$p(V) = p(a)p(t|a)p(s)p(l|s)p(b|s)p(e|t,l)p(d|e,b)p(x|e).$$

All variables are binary with levels “yes”, ”no”.

The building blocks $p(z_v|z_{pa(v)})$, for example

$$p(e|t,l)$$

are represented as multidimensional arrays (here a $2 \times 2 \times 2$ array).

In real world applications, such arrays can become very large and will often contain many zeros.

3 The curse of dimensionality

In principle (and in practice in this small toy example) we can find e.g. $p(b|a^+, d^+)$ by brute force calculations.

Recall: We have a collection of conditional probability tables (CPTs) of the form $p(v|pa(v))$:

$$\{p(a), p(t|a), p(s), p(l|s), p(b|s), p(e|t,l), p(d|e,b), p(x|e)\}$$

Brute force computations:

1) Form the joint distribution $p(V)$ by multiplying the CPTs

$$p(V) \leftarrow p(a)p(t|a)p(s)p(l|s)p(b|s)p(e|t,l)p(d|e,b)p(x|e).$$

This gives $p(V)$ represented by a table with giving a table with $2^8 = 256$ entries.

2) Find the marginal distribution $p(a, b, d)$ by marginalizing $p(V) = p(a, t, s, k, e, b, x, d)$

$$p(a, b, d) = \sum_{t,s,k,e,b,x} p(t, s, k, e, b, x, d)$$

This is table with $2^3 = 8$ entries.

3) Lastly notice that $p(b|a^+, d^+) \propto p(a^+, b, d^+)$.

Hence from $p(a, b, d)$ we must extract those entries consistent with $a = a^+$ and $d = d^+$ and normalize the result.

Alternatively (and easier): Set all entries not consistent with $a = a^+$ and $d = d^+$ in $p(a, b, d)$ equal to zero.

In chest clinic example the joint state space is $2^8 = 256$.

With 80 variables each with 10 levels, the joint state space is $10^{80} \approx$ the number of atoms in the universe!

Still, **gRain** has been succesfully used in a genetics network with 80.000 nodes... How can this happen?

The trick is to NOT to calculate the joint distribution

$$p(V) = p(a)p(t|a)p(s)p(l|s)p(b|s)p(e|t,l)p(d|e,b)p(x|e).$$

explicitly because that leads to working with high dimensional tables.

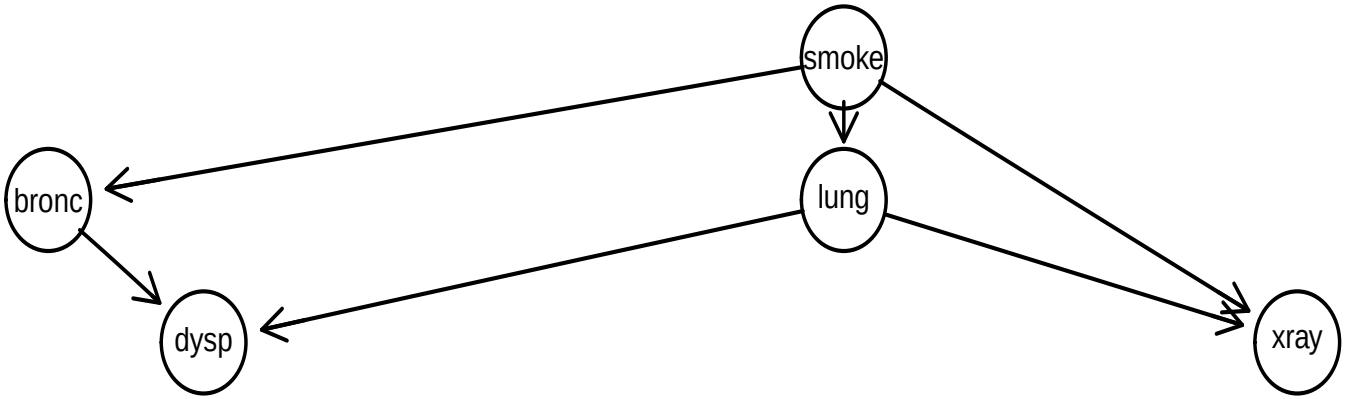
Instead we do local computations on on low dimensional tables and "send messages" between them.

The challenge is to organize these local computations. **gRain** does that for us.

4 Message passing – excerpt from chest clinic

Consider a small excerpt from the chest clinic example:

```
> library(gRbase)
> dg1 <- dag(~ smoke + bronc|smoke + lung|smoke +
              xray|smoke:lung + dysp|bronc:lung)
> plot(dg1)
```




```
> yn   <- c("yes","no")
> s    <- tabNew(~smoke, values=c(5,5),
    levels=yn, normalize="first")
> b.s  <- tabNew(~bronc | smoke, values=c(6,4,3,7),
    levels=yn, normalize="first")
> l.s  <- tabNew(~lung | smoke,  values=c(1,9,1,99),
    levels=yn, normalize="first")
> x.sl <- tabNew(~xray | smoke:lung, values=c(1,0,1,0,.1,.9,0,1),
    levels=yn, normalize="first")
```

```
> d.bl <- tabNew(~dysp | bronc:lung, values=c(9,1,7,3,8,2,1,9),
    levels=yn, normalize="first")
```

The joint pmf is the product of the cpt's

```
> p.joint <- s %a*% b.s %a*% l.s %a*% x.sl %a*% d.bl
```

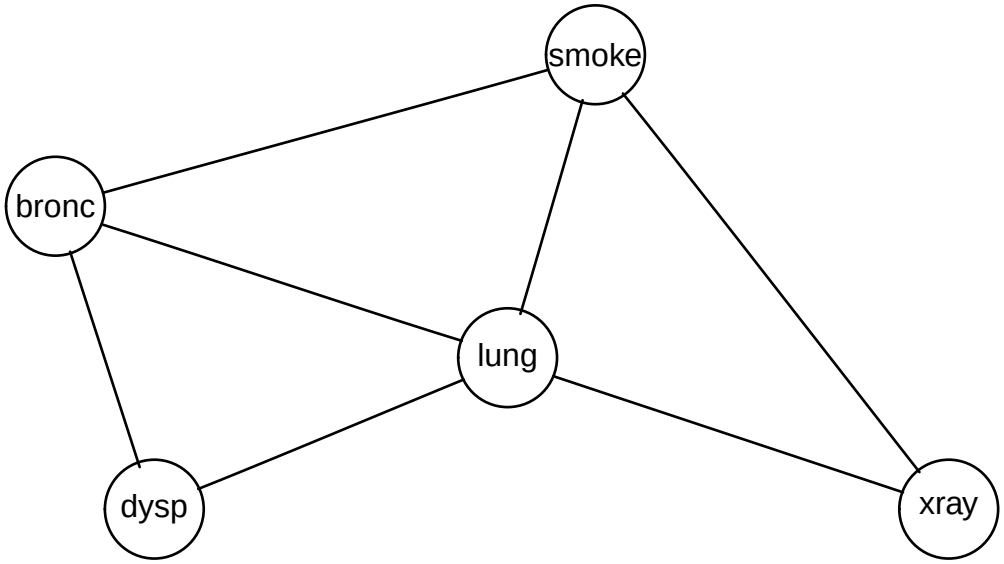
but we DO NOT want to compute the joint pmf in real-world applications. It is prohibitive in terms of storage and computing time.

4.1 Computing clique marginals

Instead we do certain local computations - outlined in the following.

First: moralize dag; then triangulate:

```
> g1 <- dg1 %>% moralize %>% triangulate
> plot(g1)
```



Next, g1 is the basis of our computations. There are three cliques (maximal complete subsets in g1):

```
> getCliques(g1)  %>% str
List of 3
 $ : chr [1:3] "lung" "bronc" "smoke"
 $ : chr [1:3] "lung" "bronc" "dysp"
 $ : chr [1:3] "lung" "xray" "smoke"
```

Form clique potentials by grouping CPTs (here we do carry out the multiplications)

```
> q1.bsl <- s %a*% b.s
> q2.dbl <- d.bl
> q3.xsl <- l.s %a*% x.sl
```

Now the clique potentials are the basis of our computations and we can forget about the CPTs (and the DAG).

Joint state-space of pmf is $2^5 = 32$. The clique potentials are smaller $3 \times 2^3 = 24$. In real-world cases the difference is much larger.

```
> q1.bsl  %>% flatten
      smoke  yes   no
bronc
yes          0.30 0.15
no           0.20 0.35
> q2.dbl  %>% flatten
     bronc yes      no
     lung  yes  no yes  no
dysp
yes         0.9 0.8 0.7 0.1
no          0.1 0.2 0.3 0.9
> q3.xsl  %>% flatten
     smoke  yes        no
     lung   yes   no  yes   no
xray
yes         0.10 0.09 0.01 0.00
no          0.00 0.81 0.00 0.99
```

The trick in message passing: Manipulate the qs such that they end up containing corresponding distributions.

```
> # From q3 to q1:
> # Marginalize "onto" (s,l):
> # Divide q3
> # Multiply q1 - does not change domain of q1
> q3.sl  <- q3.xsl %amarg% ~smoke + lung
> q3.xsl <- q3.xsl %a/0% q3.sl
> q1.bsl <- q1.bsl %a*% q3.sl
> q1.bsl %>% flatten
      smoke    yes            no
      lung     yes     no     yes     no
bronc
yes          0.0300 0.2700 0.0015 0.1485
no           0.0200 0.1800 0.0035 0.3465
```

```
> # From q1 to q2:
> # Work on q1: marginalize "onto" (b, l)
> # Divide q1
> # Multiply q2 - does not change domain of q2
> q1.bl  <- q1.bsl %amarg% ~bronc + lung
> q1.bsl <- q1.bsl %a/0% q1.bl
> q2.dbl <- q2.dbl %a*% q1.bl
> q2.dbl %>% flatten
     bronc    yes             no
     lung     yes     no      yes     no
dysp
yes         0.0284 0.3348 0.0165 0.0527
no          0.0032 0.0837 0.0071 0.4738
> q2.dbl %>% sum # check!
[1] 1
```

Now go the other way:

```
> # From q2 to q1:
> # Work on q2; marginalize onto (b,l);
> # Multiply result onto q1 - does not change domain of q1
> q2.bl  <- q2.dbl %amarg% ~bronc + lung
> q1.bsl <- q1.bsl %a*% q2.bl
> q1.bsl %>% flatten
      bronc    yes             no
      lung     yes     no      yes     no
smoke
yes          0.0300 0.2700 0.0200 0.1800
no           0.0015 0.1485 0.0035 0.3465
> q1.bsl %>% sum
[1] 1
```

```
> # From q1 to q3:
> # Work on q1; marginalize onto (s,l);
> # Multiply result onto q3 - does not change domain of q3
> q1.sl <- q1.bsl %amarg% ~smoke + lung
> q3.xsl <- q3.xsl %a*% q1.sl
> q3.xsl %>% flatten
     smoke   yes           no
     lung    yes    no   yes    no
xray
yes         0.050 0.045 0.005 0.000
no          0.000 0.405 0.000 0.495
> q3.xsl %>% sum
[1] 1
```

Empirical proof about clique marginals: In this toy example we can compute the joint pmf:

```
> p.joint <- s %a*% b.s %a*% l.s %a*% x.sl %a*% d.bl
```

Next marginalize and compare with clique potentials:

```
> p1.bsl <- p.joint %amarg% ~bronc + smoke + lung
> p2.dbl <- p.joint %amarg% ~dysp + bronc + lung
> p3.xsl <- p.joint %amarg% ~xray + smoke + lung
```

Are ps and qs identical?

```
> p1.bsl %a==% q1.bsl
[1] TRUE
> p2.dbl %a==% q2.dbl
[1] TRUE
> p3.xsl %a==% q3.xsl
[1] TRUE
```

4.2 Conditioning

Conditioning is trivial:

Suppose `xray='yes'`.

1. Multiply entries corresponding to `xray='no'` in the qs by 0.
2. Repeat all steps above. Afterwards clique potentials will contain conditionl distributions.

```
> q1.bsl <- s %a*% b.s
> q2.dbl <- d.bl
> q3.xsl <- l.s %a*% x.sl
```

```
> q3.xsl %>% flatten
     smoke  yes         no
     lung   yes   no  yes   no
xray
yes        0.10 0.09 0.01 0.00
no         0.00 0.81 0.00 0.99
> q3.xsl <- q3.xsl %aslice*% list(xray="yes")
> q3.xsl %>% flatten
     smoke  yes         no
     lung   yes   no  yes   no
xray
yes        0.10 0.09 0.01 0.00
no         0.00 0.00 0.00 0.00
```

Now repeat all steps above:

```
> q3.sl  <- q3.xsl %amarg% ~smoke + lung
> q3.xsl <- q3.xsl %a/0% q3.sl
> q1.bsl <- q1.bsl %a*% q3.sl
> q1.bsl %>% flatten
      smoke    yes             no
      lung     yes     no    yes     no
bronc
yes          0.0300 0.0270 0.0015 0.0000
no           0.0200 0.0180 0.0035 0.0000
> q1.bl  <- q1.bsl %amarg% ~bronc + lung
> q1.bsl <- q1.bsl %a/0% q1.bl
> q2.dbl <- q2.dbl %a*% q1.bl
> q2.dbl %>% flatten
     bronc    yes             no
     lung     yes     no    yes     no
dysp
yes         0.0284 0.0216 0.0165 0.0018
no          0.0032 0.0054 0.0071 0.0162
> q2.dbl <- q2.dbl / sum(q2.dbl)
> q2.dbl %>% sum # check!
[1] 1
> q2.bl  <- q2.dbl %amarg% ~bronc + lung
```

```
> q1.bsl <- q1.bsl %a*% q2.bl
> q1.bsl %>% flatten
      bronc   yes           no
      lung    yes    no   yes    no
smoke
yes         0.300 0.270 0.200 0.180
no          0.015 0.000 0.035 0.000
> q1.bsl %>% sum
[1] 1
> q1.sl <- q1.bsl %amarg% ~smoke + lung
> q3.xsl <- q3.xsl %a*% q1.sl
> q3.xsl %>% flatten
     smoke  yes         no
     lung   yes   no  yes   no
xray
yes        0.50 0.45 0.05 0.00
no         0.00 0.00 0.00 0.00
> q3.xsl %>% sum
[1] 1
```

Empirical proof

```
> p.cond <- tabDist(p.joint, cond=list(xray="yes"))
> p.cond %>% flatten
      dysp     yes                          no
      bronc    yes             no           yes             no
      lung     yes     no      yes     no   yes     no      yes     no
smoke
yes          0.2700 0.2160 0.1400 0.0180 0.0300 0.0540 0.0600 0.1620
no           0.0135 0.0000 0.0245 0.0000 0.0015 0.0000 0.0105 0.0000
```

Next marginalize and compare with clique potentials:

```
> p1.bsl <- p.cond %amarg% ~bronc + smoke + lung
> p2.dbl <- p.cond %amarg% ~dysp + bronc + lung
> p3.xsl <- p.cond %amarg% ~ smoke + lung
```

Are ps and qs identical?

```
> p1.bsl %a==% q1.bsl
[1] TRUE
> p2.dbl %a==% q2.dbl
[1] TRUE
> p3.xsl %a==% (q3.xsl %amarg% ~smoke + lung)
[1] TRUE
```

5 An introduction to the **gRain** package

5.1 Specify BN from list of CPTs

Specify chest clinic network. Can be done in many ways; one is from a list of CPTs:

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

```
> cpt.list <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
> cpt.list
cpt_spec with probabilities:
 P( asia )
 P( tub | asia )
 P( smoke )
 P( lung | smoke )
 P( bronc | smoke )
 P( either | lung tub )
 P( xray | either )
 P( dysp | bronc either )
```

```
> cpt.list$asia
asia
 yes   no
0.01 0.99
> cpt.list$tub
     asia
tub    yes   no
  yes 0.05 0.01
  no  0.95 0.99
> ftable(cpt.list$either, row.vars=1) # Notice: logical variable
       lung yes     no
       tub  yes no yes no
either
yes             1   1   1  0
no              0   0   0  1
```

5.2 Building network

```
> # Create network from CPT list:
> bn  <- grain(cpt.list)
> # Compile network (details follow)
> bn <- compile(bn)
> bn
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray"
```

5.3 Querying the network

```
> # Query network to find marginal probabilities of diseases
> disease <- c("tub", "lung", "bronc")
> bn %>% qgrain(nodes=disease)
$tub
tub
 yes   no
0.01 0.99

$lung
lung
  yes    no
0.055 0.945

$bronc
bronc
 yes   no
0.45 0.55
```

5.4 Setting evidence

```
> # Set evidence and query network again
> bn.ev <- bn %>% setEvidence(evi=list(asia="yes", dysp="yes"))
> bn.ev %>% qgrain(nodes=disease)
$tub
tub
  yes    no
0.088 0.912

$lung
lung
yes  no
0.1 0.9

$bronc
bronc
 yes   no
0.81 0.19
```

```
> # Get the evidence
> getEvidence(bn.ev)
  nodes is.hard.evidence hard.state
1  asia              TRUE        yes
2  dysp              TRUE        yes
> # Probability of observing the evidence (the normalizing constant)
> pEvidence(bn.ev)
[1] 0.0045
```

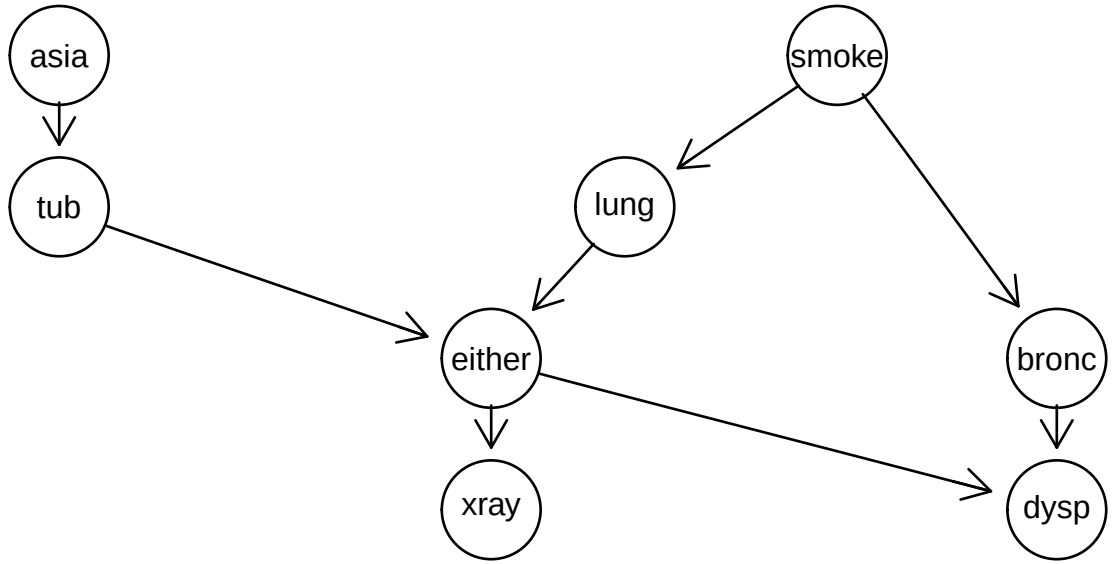
A little shortcut: Most uses of **gRain** involves 1) setting evidence into a network and 2) querying nodes. This can be done in one step:

```
> qgrain(bn,
    evidence=list(asia="yes", dysp="yes"),
    nodes=disease)
$tub
tub
  yes    no
0.088 0.912

$lung
lung
yes  no
0.1 0.9

$bronc
bronc
 yes   no
0.81 0.19
```

5.5 Specify BN from DAG and data**

If the structure of the DAG is known and we have data, we can just do:

```
> vpa <- list("asia", c("tub", "asia"), "smoke",
    c("lung", "smoke"), c("bronc","smoke"),
    c("either", "lung", "tub"),
    c("xray", "either"), c("dysp", "bronc", "either"))
> dg <- dag( vpa )
> plot(dg)
```

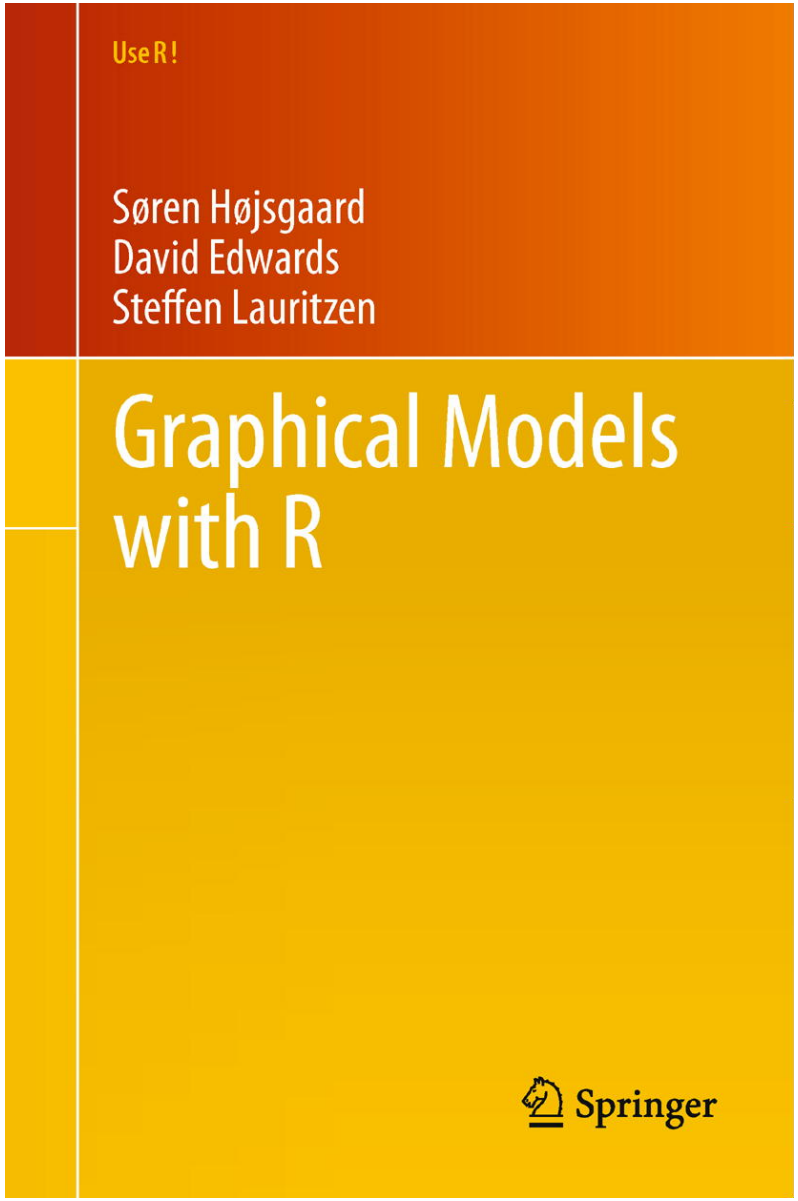


```
> data(chestSim1000, package="gRbase")
> head(chestSim1000)
  asia tub smoke lung bronc either xray dysp
1   no  no    no   no   yes     no   no  yes
2   no  no   yes   no   yes     no   no  yes
3   no  no   yes   no    no     no   no   no
4   no  no    no   no    no     no   no   no
5   no  no   yes   no   yes     no   no  yes
6   no  no   yes  yes   yes    yes  yes  yes
> bn2 <- grain(dg, data=chestSim1000, smooth=.1)
> bn2
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray"
```

The CPTs are estimated as the relative frequencies.

6 Wrapping up

6.1 Book: Graphical Models with R



6.2 Package versions

We shall in this tutorial use the R–packages **gRbase**, **gRain** and **gRim**.

Go to `http://people.math.aau.dk/~sorenh/software/gR` for installation instructions.

The tutorial is based on these versions of the packages (which are available on github):

```
> packageVersion("gRbase")
[1] '1.8.6.9001'
> packageVersion("gRain")
[1] '1.3.6'
> packageVersion("gRim")
[1] '0.2.5'
```