Graphical Models and Bayesian Networks

Short course; Zürich, Switzerland, November 3.+4. 2016

REVISED - February 2017

February 5, 2017

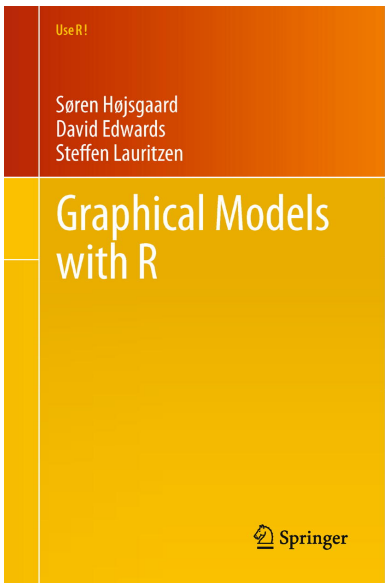
Søren Højsgaard
Department of Mathematical Sciences
Aalborg University, Denmark
http://people.math.aau.dk/~sorenh/

Contents

1 Outline

- Bayesian networks and the **gRain** package
- Probability propagation; conditional independence restrictions and dependency graphs
- Learning structure with log–linear, graphical and decomposable models for contingency tables
- Using the **gRim** package for structural learning.
- Convert decomposable model to Bayesian network.
- Other packages for structure learning.

1.1 Book: Graphical Models with R



The book, written by some of the people who laid the foundations of work in this area, would be ideal for researchers who had read up on the theory of graphical models and who wanted to apply them in practice. It would also make excellent supplementary material to accompany a course text on graphical modelling. I shall certainly be recommending it for use in that role...the book is neither a text on graphical models nor a manual for the various packages, but rather has the more modest aims of introducing the ideas of graphical modelling and the capabilities of some of the most important packages. It succeeds admirably in these aims. The

simplicity of the commands of the packages it uses to illustrate is apparent, as is the power of the tools available.

International Statistical Review, Volume 31, Issue 2 review by David J. Hand

1.2 Package versions

We shall in this tutorial use the R–packages **gRbase**, **gRain** and **gRim**.
Installation: First install bioconductor packages

```
source("http://bioconductor.org/biocLite.R");
biocLite(c("graph","RBGL","Rgraphviz"))
```

Then install **gRbase**, **gRain** and **gRim** from CRAN

```
install.packages("gRbase", dependencies=TRUE)
install.packages("gRain", dependencies=TRUE)
install.packages("gRim", dependencies=TRUE)
```

Go to `http://people.math.aau.dk/~sorenh/software/gR` and locate the development versions (on which this tutorial is based).

Alternatively, go to `https://github.com/hojsgaard` and find development versions.

```
packageVersion("gRbase")

## [1] '1.8.2'

packageVersion("gRain")

## [1] '1.3.1'

packageVersion("gRim")

## [1] '0.1.17'
```

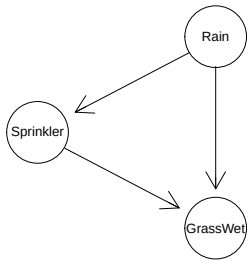
```
library(gRain)
library(Rgraphviz)
library(gRim)
```

2 Example: The sprinkler network

2.1 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.

Think about R, S and G as discrete random variables (could write X_R, X_S, X_G but that is too cumbersome).

Suppose we have a joint probability mass function (pmf) $p_{GSR}(g, s, r)$.

Using Bayes' formula twice, $p_{GSR}(g, s, r)$ can be factorized as

$$p_{GSR}(g, s, r) = p_{G|SR}(g|s, r)p_{S|R}(s|r)p_R(r)$$

We shall allow the more informal notation

$$P(G, S, R) = P(G|S, R)P(S, R) = P(G|S, R)P(S|R)P(R)$$

In the context of building Bayesian networks, we start in "the other end" by specifying the terms

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

We call these terms *conditional probability tables* (or *CPT*s).

In `R`, CPTs are simply arrays.

Then we construct a joint pmf by

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

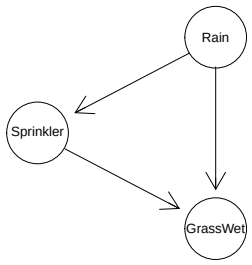
Notice this: Terms on the right hand of

$$P(G, S, R) = P(G|S, R)P(S|R)P(R)$$

has the form

$$p(v|\,\mathrm{pa}(v))$$

relative to the DAG (directed acyclic graph).



2.2 Conditional probability tables (CPTs)

Conditional probability tables (CPTs) are in R as arrays named dimnames. For compact printing of arrays define utility function

```
flat <- function(x){ftable(x, row.vars=1)} ## Just a utility
```

Arrays can be created e.g. with `array()` or as follows:

```
yn <- c("yes", "no")
p.R <- array( c(.2, .8), dim = 2,
              dimnames = list(Rain=yn) )
p.S_R <- array(c(.01, .99, .4, .6), dim=c(2, 2),
               dimnames=list(Sprinkler=yn, Rain=yn))
p.G_SR <- array(c(.99, .01, .8, .2, .9, .1, 0, 1),
                dim = c(2, 2, 2),
                dimnames=list(GrassWet=yn, Sprinkler=yn, Rain=yn))
```

```
p.R
```

```
## Rain
## yes  no
## 0.2 0.8
```

```
p.S_R %>% flat
```

```
##           Rain  yes   no
## Sprinkler
## yes           0.01 0.40
## no            0.99 0.60
```

```
p.G_SR  %>% flat
```

```
##          Sprinkler  yes        no
##          Rain       yes   no  yes   no
## GrassWet
## yes                0.99 0.90 0.80 0.00
## no                 0.01 0.10 0.20 1.00
```

An alternative is parray():

```
yn <- c("yes", "no")
ssp <- list(Rain = yn, Sprinkler = yn, GrassWet = yn)
p.R <- parray("Rain", levels=ssp, values=c(.2, .8))
p.S_R <- parray(c("Sprinkler", "Rain"), levels = ssp,
              values=c(.01, .99, .4, .6))
p.G_SR <- parray(~ GrassWet:Sprinkler:Rain, levels = ssp,
               values=c(.99, .01, .8, .2, .9, .1, 0, 1))
```

```
p.R
```

```
## Rain
## yes  no
## 0.2 0.8
## attr(,"class")
## [1] "parray" "array"
```

```
p.S_R %>% flat
```

```
##           Rain  yes   no
## Sprinkler
## yes           0.01 0.40
## no            0.99 0.60
```

```
p.G_SR  %>% flat

##          Sprinkler  yes        no
##          Rain       yes   no   yes   no
## GrassWet
## yes                 0.99 0.90 0.80 0.00
## no                  0.01 0.10 0.20 1.00
```

An alternative (syntax may change in the future) is ar_new():

```
yn <- c("yes", "no")
ssp <- list(Rain = yn, Sprinkler = yn, GrassWet = yn)
p.R <- ar_new("Rain", levels=ssp, values=c(.2, .8))
p.S_R <- ar_new(c("Sprinkler", "Rain"), levels = ssp,
              values=c(.01, .99, .4, .6))
p.G_SR <- ar_new(~ GrassWet:Sprinkler:Rain, levels = ssp,
               values=c(.99, .01, .8, .2, .9, .1, 0, 1))
```

2.3 A small digression: Operations on arrays

```
T1 <- ar_new( ~a:b, levels=c(2, 2), values=1:4 )
T2 <- ar_new( ~b:c, levels=c(2, 2), values=5:8 )
T1 %>% flat

##    b b1 b2
## a
## a1    1  3
## a2    2  4

T2 %>% flat

##    c c1 c2
## b
## 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 $T = T_1T_2$ is a function of (a, b, c) defined as

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

Similar definitions of $T_1 + T_2$, $T_1 - T_2$, T_1/T_2.

```
ar_prod( T1, T2 ) %>% flat ## multiple arguments like prod()

##    c c1    c2
##    a a1 a2 a1 a2
## b
## b1    5 10  7 14
## b2   18 24 24 32

ar_sum(T1, T2) %>% flat    ## multiple arguments like sum()

##    c c1    c2
##    a a1 a2 a1 a2
## b
## b1    6  7  8  9
## b2    9 10 11 12
```

Binary operators

```
ar_mult(T1, T2)
ar_div(T1, T2)
ar_add(T1, T2)
ar_subt(T1, T2)
T1 %a*% T2
T1 %a/% T2
T1 %a+% T2
T1 %a+% T2
```

2.4 Joint pmf

Joint pmf:

```
## P(G,S,R)
p.GSR <- ar_prod( p.G_SR, p.S_R, p.R )
p.GSR  %>% flat

##      GrassWet      yes              no
##      Sprinkler     yes      no      yes      no
## Rain
## yes            0.00198 0.15840 0.00002 0.03960
## no             0.28800 0.00000 0.03200 0.48000

sum( p.GSR ) # check

## [1] 1
```

2.5 Using Bayes' formula

Question: What is the probability that it is raining given that the grass is wet?
Answer: Use Bayes formula for marginalization and conditioning:

$$
\begin{aligned}
P(G,R) &= \sum_{S=y,n} P(G,S,R) \\
P(R|G=y) &= \frac{P(G=y,R)}{P(G=y)} \\
&= \frac{P(R,G=y)}{\sum_{R=y,n} P(R,S,G=y)}
\end{aligned}
$$

This question - and others - can be answered with `ar_dist()`:

```
ar_dist(p.GSR, marg="Rain", cond=list(GrassWet="yes"))

## Rain
##   yes    no
## 0.358 0.642
```

For example; what is $P(R,S|G=y)$:

```
ar_dist(p.GSR, cond=list(GrassWet="yes"))

##      Sprinkler
## Rain      yes    no
##   yes 0.00442 0.353
##   no  0.64231 0.000
```

In detail we have:

```
## Marginalize -> P(R,G)
p.RG  <- ar_marg(p.GSR, c("Rain", "GrassWet")); p.RG

##      GrassWet
## Rain    yes     no
##   yes 0.160 0.0396
##   no  0.288 0.5120

## Marginalize -> P(G)
p.G   <- ar_marg(p.RG, "GrassWet"); p.G
```

```
## GrassWet
##   yes    no
## 0.448 0.552


## Condition -> P(R|G)
p.R_G <- ar_div(p.RG, p.G); p.R_G


##         Rain
## GrassWet    yes    no
##      yes 0.3577 0.642
##      no  0.0718 0.928


## Pick the slice -> P(R|G=yes)
ar_slice( p.R_G, slice=list(GrassWet="yes"))


##   yes    no
## 0.358 0.642
```

2.6 The curse of dimensionality

In the example, the joint state space is $2^3 = 8$.

We calculated the joint pmf (a $2 \times 2 \times 2$ table) by multiplying conditionals, then we marginalized and then we conditioned.

With 80 variables each with 10 levels, the joint state space is 10^{80}.

$10^{80} \approx$ the number of atoms in the universe.

Fortunately, there is often a structure to the model such that one need NOT calculate the full joint pmf.

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

This structure arise from conditional independence restrictions.

gRain exploits this structure and has been used succesfully on networks with 10.000s of variables.

2.7 A small digression: Creating and displaying graphs

Default representation of DAG is as a *graphNEL* object

```
dg <- dag(~ Rain + Sprinkler|Rain + GrassWet|Rain:Sprinkler)
dg
```

```
## A graphNEL graph with directed edges
## Number of Nodes = 3
## Number of Edges = 3
```

```
nodes(dg)
```

```
## [1] "Rain"      "Sprinkler" "GrassWet"
```

```
edges(dg)
```

```
## $Rain
## [1] "Sprinkler" "GrassWet"
##
## $Sprinkler
## [1] "GrassWet"
##
## $GrassWet
## character(0)
```

```
edgeList(dg)
```

```
## [[1]]
## [1] "Rain"      "Sprinkler"
##
## [[2]]
## [1] "Rain"     "GrassWet"
##
## [[3]]
## [1] "Sprinkler" "GrassWet"
```

Another representation is as an *adjacency matrix*; either a *dense* or a *sparse* matrix:

```
as(dg, "matrix")
```

```
##           Rain Sprinkler GrassWet
## Rain         0         1        1
## Sprinkler    0         0        1
## GrassWet     0         0        0
```

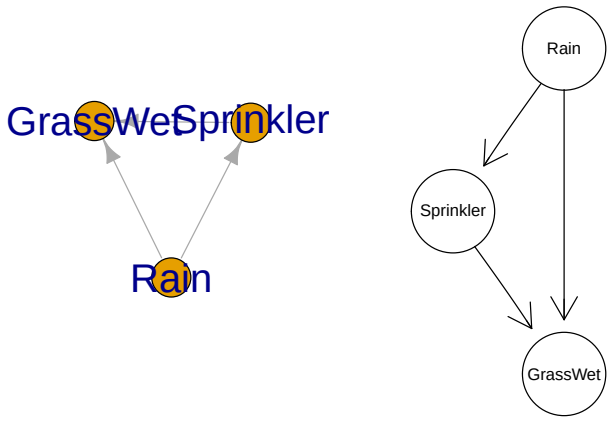
```
as(dg, "dgCMatrix")
```

```
## 3 x 3 sparse Matrix of class "dgCMatrix"
##           Rain Sprinkler GrassWet
## Rain         .         1        1
## Sprinkler    .         .        1
## GrassWet     .         .        .
```

To display graphs there are two options in these packages: 1) Using the `iplot()` method which is based on the **igraph** package; 2) using the `plot()` method which is based on the **Rgraphviz** package.

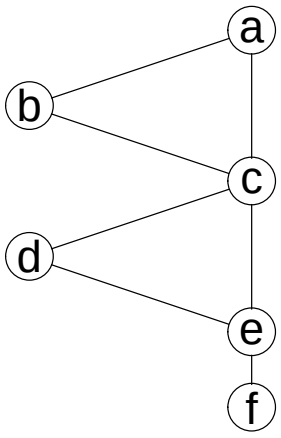
The **Rgraphviz** package must be loaded manually by the user.

```
dg <- dag(~ Rain + Sprinkler|Rain + GrassWet|Rain:Sprinkler)
par(mfrow=c(1,2))
iplot( dg )
library(Rgraphviz)
plot( dg )
```



Undirected graphs can be created as

```
ugr <- ug(~ a:b:c + c:d:e + e:f)
plot(ugr)
```



3 Conditional independence

Conditional independence restrictions are essential in Bayesian networks and graphical models.

Let X, Y, Z be random vectors with joint pdf or pmf $f_{XYZ}(x, y, z)$.

The statement that X and Y are conditionally independent given Z, written $X \perp\!\!\!\perp Y|Z$, means that X and Y are independent in the conditional distribution given given $Z = z$ for each possible value z of Z.

In terms of a joint density we have

$$f_{XY|Z}(x, y|z) = f_{X|Z}(x|z)f_{Y|Z}(y|z)$$

or equivalently that

$$f_{X|YZ}(x|y, z) = f_{X|Z}(x|z)$$

Once we know z we will obtain no additional information about x by also getting to know y.

Independence is a "special case" of conditional independence where we need not "condition on anything": $X \perp\!\!\!\perp Y$ iff

$$f_{XY}(x, y) = f_X(x)f_Y(y)$$

or - equivalently -

$$f_{X|Y}(x \,|\, y) = f_X(x)$$

In practice it is often easiest to check conditional independence using the factorization criterion: If

$$f_{XYZ}(x, y, z) = q_1(x, z)q_2(y, z)$$

for non–negative functions $q_1()$ and $q_2()$ then $X \perp\!\!\!\perp Y|Z$.

Example 3.1 Suppose $X = (X_1, X_2, X_3) \sim N_3(0, \Sigma)$ where $K = \Sigma^{-1}$ has the form

$$K = \begin{bmatrix} k_{11} & 0 & k_{13} \\ 0 & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}$$

Then $X_1 \perp\!\!\!\perp X_2|X_3$. (We call K the *concentration matrix*.)

```
K <- matrix(c(1,0,.5, 0,1,.5, .5,.5,1), nr=3); K
```

```
##      [,1] [,2] [,3]
## [1,]  1.0  0.0  0.5
## [2,]  0.0  1.0  0.5
## [3,]  0.5  0.5  1.0
```

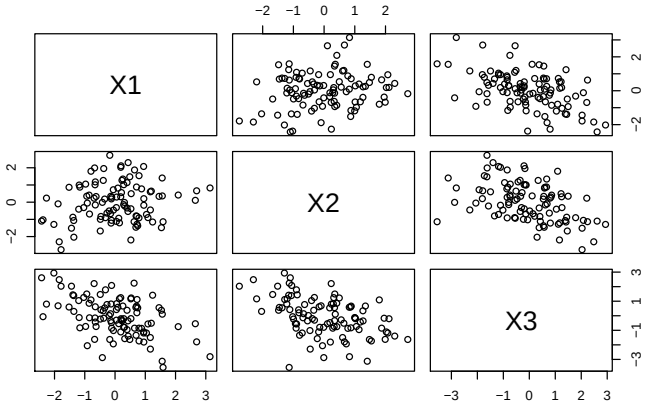
```
S <- solve(K); S
```

```
##      [,1] [,2] [,3]
## [1,]  1.5  0.5   -1
## [2,]  0.5  1.5   -1
## [3,] -1.0 -1.0    2
```

```
n <- 100
set.seed(12)
d <- data.frame(MASS::mvrnorm(n, mu=c(0,0,0), Sigma=S))
```

Marginally, all variables are correlated:

```
pairs(d)
```




```
cor(d)
```

```
##         X1     X2     X3
## X1  1.000  0.179 -0.503
## X2  0.179  1.000 -0.458
## X3 -0.503 -0.458  1.000
```

Conditional independence here means that in the conditional distribution of X_3, X_1 and X_2 are independent. If $A = \{1, 2\}$ and $B = \{3\}$ then the conditional covariance is

$$\Sigma_{A \cdot B} = \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}$$

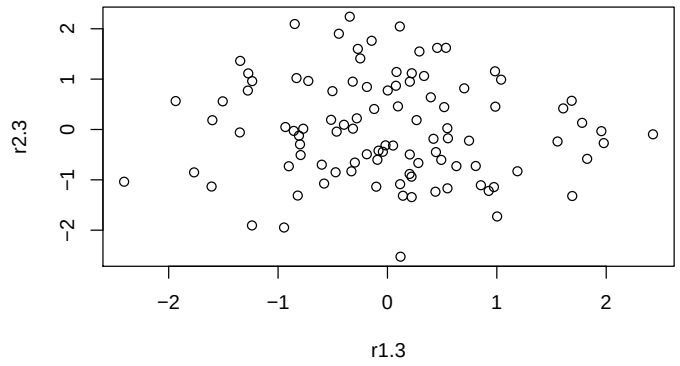
which gives:

```
S[1:2, 1:2] - S[1:2, 3, drop=F] %*% solve(S[3,3, drop=F], S[3,1:2, drop=F])
```

```
##       [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Another view is as follows: Regress X1 and X2 on X3 and look at the residuals

```
r1.3 <- resid( lm( X1 ~ X3, data=d) )
r2.3 <- resid( lm( X2 ~ X3, data=d) )
plot(r1.3, r2.3)
```




```
cor(r1.3, r2.3)

## [1] -0.0661

cov( cbind(r1.3, r2.3) )

##         r1.3    r2.3
## r1.3  0.8644 -0.0617
## r2.3 -0.0617  1.0080

cor( cbind(r1.3, r2.3) )

##         r1.3    r2.3
## r1.3  1.0000 -0.0661
## r2.3 -0.0661  1.0000
```

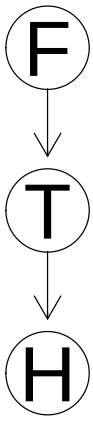
The residuals are now uncorrelated so after adjusting for X3, any association between X1 and X2 has been removed. An alternative view is this: Regress X1 on X2 and X3:

```
coef( summary( lm( X1 ~ X2 + X3, data=d) ) )

##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -0.000161     0.0938 -0.00172 9.99e-01
## X2          -0.061215     0.0938 -0.65248 5.16e-01
## X3          -0.437369     0.0809 -5.40856 4.57e-07
```

Cleary X1 does not depend on X2 once we know X3. □

3.1 Exercises - flu; fever; headache

```
plot(dag(~ F + T|F + H|T))
```



A random person has flu (F) with probability 0.01. A person with flu will almost certainly have fever / a raised body temperature (T): A person with flu has fever with probability 0.99 while a random person without flu will have fever with probability 0.001. A person with fever has headache (H) with probability 0.7 while a person without fever has headache with probabilty 0.05

This sitation shouls be depicted in the graph above.

1. What is the probability (i) that a random person has flu, (ii) that a random person has fever, (iii) that a random person has headache?
2. What is the probability that a person has headache given that he/she has flu?
3. What is the probability that a random person has flu given that he/she has headache?
4. What is the probability that a random person has flu given that he/she has fever?
5. What is the probability that a random person has flu given that he/she has fever and headache? Compare with your results above. Conclusion?

3.2 Exercises – a lecturers life

This example is taken from
`http://www.csse.monash.edu.au/bai/book/BAI_Chapter2.pdf`:

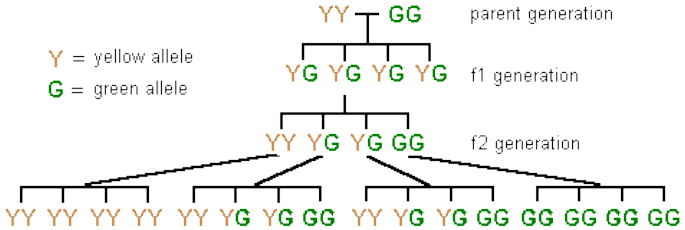
A Lecturers Life: Dr. Ann Nicholson spends 60% of her work time in her office. The rest of her work time is spent elsewhere. When Ann is in her office, half the time her light is off (when she is trying to hide from students and get research done). When she is not in her office, she leaves her light on only 5% of the time. 80% of the time she is in her office, Ann is logged onto the computer. Because she sometimes logs onto the computer from home, 10% of the time she is not in her office, she is still logged onto the computer.

1. Construct a Bayesian network to represent the Lecturers Life scenario just described.
2. Suppose a student checks Dr. Nicholsons login status and sees that she is logged on. What effect does this have on the students belief that Dr. Nicholsons light is on?

4 Example: Mendelian segregation

4.1 Mendel's Genetics

A very clear introduction at:
`http://anthro.palomar.edu/mendel/mendel_1.htm`



A pea will have two alleles related to its color. An *allele* can be y or g. A pea recieves one allele from each parent.

The *genotype* is an unordered pair of alleles: $\{y, y\}$, $\{y, g\}$ or $\{g, g\}$. Think of genotype as a random variable with states $\{yy, yg, gg\}$.

```
gts <- c("yy", "yg", "gg")
```

The alleles combine independently and therefore the probability of each genotype is

```
gtprobs <- c(.25, .50, .25)
```

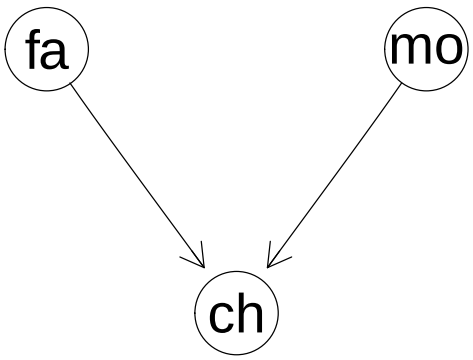
The *phenotype* is what can be observed, i.e. wheter the pea is yellow or green.

The y–allele is *dominant*; the g–allele is *recessive*: The pea will be green (the phenotype) only if the genotype is gg; if the allele is yy or yg, the pea will be yellow.

Therefore yellow and gree peas will appear in the proportions $3 : 1$.

Peas do not have fathers and mothers, but only parents - but let us for later purposes think of them as fathers and mothers.

```
dG <- dag(~ ch|fa:mo)
plot( dG )
```




```
men <- mendel( c("y", "g"), names = c("ch", "fa", "mo") )
head( men, 10 )
```

```
##    ch fa mo prob
## 1  yy yy yy  1.0
## 2  yg yy yy  0.0
```

```
## 3  gg yy yy  0.0
## 4  yy yg yy  0.5
## 5  yg yg yy  0.5
## 6  gg yg yy  0.0
## 7  yy gg yy  0.0
## 8  yg gg yy  1.0
## 9  gg gg yy  0.0
## 10 yy yy yg  0.5
```

```
dim( men )
```

```
## [1] 27  4
```

```
## For later use, save inheritance probabilities
inheritance <- men$prob
head( inheritance )
```

```
## [1] 1.0 0.0 0.0 0.5 0.5 0.0
```

Conditional distribution $p(\texttt{ch} \mid \texttt{fa}, \texttt{mo})$:

```
ssp <- list(fa = gts, mo = gts, ch = gts)
p.c_fm <- ar_new(~ ch:fa:mo, levels=ssp, values=inheritance)
ftable( p.c_fm, row.vars = "ch" )
```

```
##    fa   yy             yg             gg
##    mo   yy   yg   gg   yy   yg   gg   yy   yg   gg
## ch
## yy    1.00 0.50 0.00 0.50 0.25 0.00 0.00 0.00 0.00
## yg    0.00 0.50 1.00 0.50 0.50 0.50 1.00 0.50 0.00
## gg    0.00 0.00 0.00 0.00 0.25 0.50 0.00 0.50 1.00
```

In equilibrium the population frequencies of the alleles will be

```
gts
```

```
## [1] "yy" "yg" "gg"
```

```
gtprobs
```

```
## [1] 0.25 0.50 0.25
```

```
p.fa <- ar_new(~ fa, levels=ssp, values=gtprobs); p.fa
```

```
## fa
##   yy   yg   gg
## 0.25 0.50 0.25
```

```
p.mo <- ar_new(~ mo, levels=ssp, values=gtprobs); p.mo
```

```
## mo
##   yy   yg   gg
## 0.25 0.50 0.25
```

Joint distribution is $p(ch, fa, mo) = (ch|fa, mo)p(fa)p(mo)$:

```
joint <- ar_prod( p.fa, p.mo, p.c_fm )
ftable(round( joint, 3) , row.vars="ch")
```

```
##    fa    yy                yg                gg
##    mo    yy    yg    gg    yy    yg    gg    yy    yg    gg
## ch
## yy    0.062 0.062 0.000 0.062 0.062 0.000 0.000 0.000 0.000
## yg    0.000 0.062 0.062 0.062 0.125 0.062 0.062 0.062 0.000
## gg    0.000 0.000 0.000 0.000 0.062 0.062 0.000 0.062 0.062
```

Marginal distributions:

```
## Not surprising:
ar_dist( joint, marg="fa")
```

```
## fa
##   yy   yg   gg
## 0.25 0.50 0.25
```

```
## Because of equilibrium
ar_dist( joint, marg="ch")
```

```
## ch
##   yy   yg   gg
## 0.25 0.50 0.25
```

Now condition on mother:

```
ar_dist( joint, marg="fa", cond=list(mo="yy"))
```

```
## fa
```

```
##   yy   yg   gg
## 0.25 0.50 0.25
```

```
ar_dist( joint, marg="ch", cond=list(mo="yy"))
```

```
## ch
##  yy  yg  gg
## 0.5 0.5 0.0
```

Conclusions?

Now condition on child

```
ar_dist(joint, marg="fa")
```

```
## fa
##   yy   yg   gg
## 0.25 0.50 0.25
```

```
ar_dist(joint, marg="fa", cond=list(ch="gg"))
```

```
## fa
##  yy  yg  gg
## 0.0 0.5 0.5
```

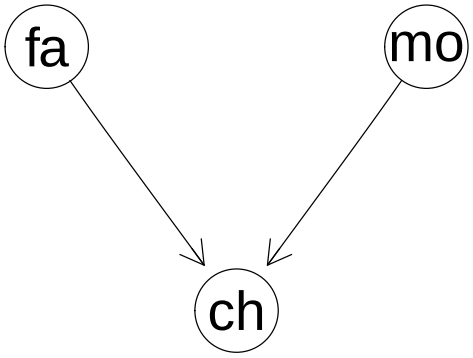
```
ar_dist(joint, marg="fa", cond=list(ch="yg"))
```

```
## fa
##   yy   yg   gg
## 0.25 0.50 0.25
```

```
ar_dist(joint, marg="fa", cond=list(ch="yg", mo="gg"))
```

```
## fa
##  yy  yg  gg
## 0.5 0.5 0.0
```

Conclusions?



Joint distribution is $p(\texttt{ch}, \texttt{fa}, \texttt{mo}) = p(\texttt{ch}|\texttt{fa}, \texttt{mo})p(\texttt{fa})p(\texttt{mo})$.

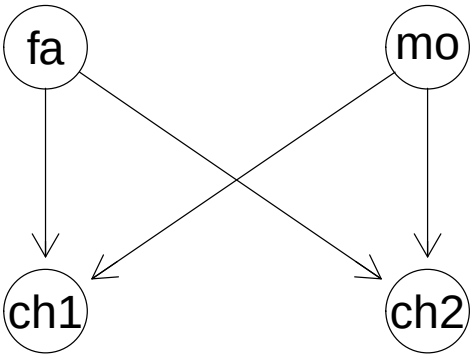
Marginally, `fa` and `mo` are independent:

$$p(\texttt{fa}, \texttt{mo}) = \sum_{\texttt{ch}} (\texttt{ch}|\texttt{fa}, \texttt{mo})p(\texttt{fa})p(\texttt{mo}) = p(\texttt{fa})p(\texttt{mo})$$

But conditionally on `ch`, `fa` and `mo` are NOT independent: We do not have a factorization:

$$p(\texttt{ch}, \texttt{fa}, \texttt{mo}) = (\texttt{ch}|\texttt{fa}, \texttt{mo})p(\texttt{fa})p(\texttt{mo}) \neq q_1(\texttt{ch}, \texttt{fa})q_2(\texttt{ch}, \texttt{mo})$$

4.2 Now with two children



Joint distribution is:

$$p(\texttt{ch1}, \texttt{ch2}, \texttt{fa}, \texttt{mo}) = p(\texttt{ch1}|\texttt{fa}, \texttt{mo})p(\texttt{ch2}|\texttt{fa}, \texttt{mo})p(\texttt{fa})p(\texttt{mo})$$

Clearly $\texttt{ch1} \perp\!\!\!\perp \texttt{ch2} \,|\, (\texttt{fa}, \texttt{mo})$ because

$$p(\texttt{ch1}, \texttt{ch2}, \texttt{fa}, \texttt{mo}) = g(\texttt{ch1}, \texttt{fa}, \texttt{mo}) h(\texttt{ch2}, \texttt{fa}, \texttt{mo})$$

```
ssp <- list(fa = gts, mo = gts, ch = gts, ch1 = gts, ch2 = gts)
head( inheritance )
```

```
## [1] 1.0 0.0 0.0 0.5 0.5 0.0
```

```
p.c1_fm <- ar_new(~ ch1:fa:mo, levels=ssp, values=inheritance)
p.c2_fm <- ar_new(~ ch2:fa:mo, levels=ssp, values=inheritance)

joint <- ar_prod(p.fa, p.mo, p.c1_fm, p.c2_fm)
joint  %>% round(2)  %>% ftable(col.vars = c("ch1", "ch2"))
```

```
##       ch1   yy             yg             gg
##       ch2   yy   yg   gg   yy   yg   gg   yy   yg   gg
## fa mo
## yy yy     0.06 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
##    yg     0.03 0.03 0.00 0.03 0.03 0.00 0.00 0.00 0.00
##    gg     0.00 0.00 0.00 0.00 0.06 0.00 0.00 0.00 0.00
## yg yy     0.03 0.03 0.00 0.03 0.03 0.00 0.00 0.00 0.00
##    yg     0.02 0.03 0.02 0.03 0.06 0.03 0.02 0.03 0.02
##    gg     0.00 0.00 0.00 0.00 0.03 0.03 0.00 0.03 0.03
## gg yy     0.00 0.00 0.00 0.00 0.06 0.00 0.00 0.00 0.00
##    yg     0.00 0.00 0.00 0.00 0.03 0.03 0.00 0.03 0.03
##    gg     0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.06
```

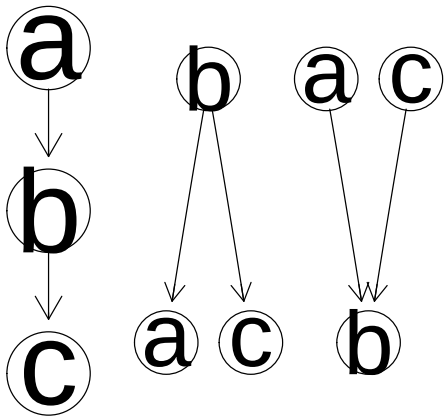
4.3 Exercises

1. On your own computer, construct the 4–way table above.
2. What is $p(\texttt{ch1}|\texttt{ch2} = yy)$?
3. What is $p(\texttt{ch1}|\texttt{ch2} = yy, \texttt{fa} = yy)$?
4. What is $p(\texttt{ch1}|\texttt{ch2} = yy, \texttt{fa} = yy, \texttt{mo} = yg)$?
5. What is $p(\texttt{ch1}|\texttt{fa} = yy, \texttt{mo} = yg)$?

Hint: Your friend is `ar_dist`

4.4 A small digression: DAG configurations and conditional independence

```
par(mfrow=c(1,3))
plot( dag(~ a + b|a + c|b) )
plot( dag(~ b + a|b + c|b) )
plot( dag(~ b + b|a:c + c) )
```



$$p(a,b,c) = p(a)p(b|a)p(c|b)$$
$$p(a,b,c) = p(b)p(a|b)p(c|b)$$
$$p(a,b,c) = p(b)p(b|a,c)p(c)$$

Which cases gives a contitional independence restriction?

4.5 Building a network

We have seen things done by brute force computations; now we build at bayesian network:

For later purposes we shall assume that the genotype frequencies are not the Mendelian ones but:

```
gtprobs2 <- c(.09, .42, .49)
gts


## [1] "yy" "yg" "gg"
```

```
ssp
```

```
## $fa
## [1] "yy" "yg" "gg"
##
## $mo
## [1] "yy" "yg" "gg"
##
## $ch
## [1] "yy" "yg" "gg"
##
## $ch1
## [1] "yy" "yg" "gg"
##
## $ch2
## [1] "yy" "yg" "gg"
```

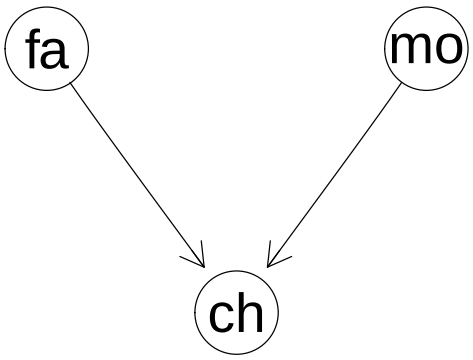
```
p.fa <- ar_new(~ fa, levels=ssp, values=gtprobs2)
p.mo <- ar_new(~ mo, levels=ssp, values=gtprobs2)
cptl <- compileCPT( list( p.fa, p.mo, p.c_fm ) ); cptl
```

```
## CPTspec with probabilities:
##  P( fa )
##  P( mo )
##  P( ch | fa mo )
```

```
trio <- grain( cptl )
trio
```

```
## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:3] "fa" "mo" "ch"
```

```
plot( trio )
```



4.6 Joint/marginal distributions

```
qgrain( trio, nodes = c("fa", "ch"),
       type="marginal" ) ## Default type


## $fa
## fa
##   yy   yg   gg
## 0.09 0.42 0.49
##
## $ch
## ch
##   yy   yg   gg
## 0.09 0.42 0.49
```

```
qgrain( trio, nodes = c("fa", "ch"), type="joint")  %>%
    ftable(row.vars="ch")


##    fa    yy    yg    gg
## ch
## yy    0.027 0.063 0.000
## yg    0.063 0.210 0.147
## gg    0.000 0.147 0.343
```

4.7 Evidence

If we observe a configuration of some of the variables, this can be entered as evidence. Then the network gives the

1. conditional distribution given the evidence
2. marginal probability of the evidence

```
## Network with evidence entered
trio_ev <- setEvi(trio, evidence=list(ch="yg", mo="yy"))
## or trio_ev <- setEvi(trio, nodes=c("ch", "mo"),
##                          states = c("yg", "yy"))
## p(father | child = yg, mother = yy)
qgrain(trio_ev, nodes="fa")


## $fa
## fa
##  yy  yg  gg
## 0.0 0.3 0.7


## Removing all entered evidence
trio_ev2 <- dropEvi(trio_ev)
## p(father)
qgrain(trio_ev2, nodes="fa")


## $fa
## fa
##   yy   yg   gg
## 0.09 0.42 0.49
```

4.8 Probability of configuration of set of variables

Method 1 (the smart way): Enter the configuration as evidence and get the normalising constant.

```
tr <- setEvi(trio, evidence = c(ch="yg", mo="yy"))
pEvidence(tr)


## [1] 0.063
```

Method 2: Get the entire joint distribution and find your configuration:

```
qgrain(trio, nodes=c("ch", "mo"), type = "joint")


##     ch
```

```
## mo      yy    yg    gg
##   yy 0.027 0.063 0.000
##   yg 0.063 0.210 0.147
##   gg 0.000 0.147 0.343
```

4.9 Simulation

We can simulate directly from the distribution the Bayesian network represents:

```
## Prior distribution
sim1 <- simulate(trio, 1000)
head(sim1, 4)


##   fa mo ch
## 1 gg gg gg
## 2 gg gg gg
## 3 gg gg gg
## 4 yg gg gg

## Posterior after observing child=yg and mother=yy
sim2 <- simulate(trio_ev, 1000)
head(sim2, 4)


##   fa mo ch
## 1 gg yy yg
## 2 yg yy yg
## 3 gg yy yg
## 4 gg yy yg
```

The frequencies do match:

```
apply(sim1, 2, table)


##     fa  mo  ch
## gg 499 497 495
## yg 404 422 417
## yy  97  81  88

apply(sim2, 2, table)


## $fa
##
```

```
##  gg  yg
## 714 286
##
## $mo
##
##   yy
## 1000
##
## $ch
##
##   yg
## 1000
```

5 Example: Paternity testing

A "mother pea" with genotype yy has a child with genotype yg.

She claims that "Pea X", who has genotype yg, is the father of her child.

The *evidence* in this case could be the observed genotypes of the mother and the child.

We compare the probability of the evidence under two alternative hypotheses:

$$H_1\text{: Pea X is the father}$$
$$\text{vs.}$$
$$H_2\text{: Some unknown pea is the father}$$

We need to compute

$$LR = \frac{Pr(ch = yg, m = yy \,|\, H_1)}{Pr(ch = yg, m = yy \,|\, H_2)} = \frac{Pr(ch = yg, m = yy \,|\, fa = yg)}{Pr(ch = yg, m = yy)}$$

$$LR = \frac{Pr(ch = yg, m = yy \,|\, fa = yg)}{Pr(ch = yg, m = yy)}$$

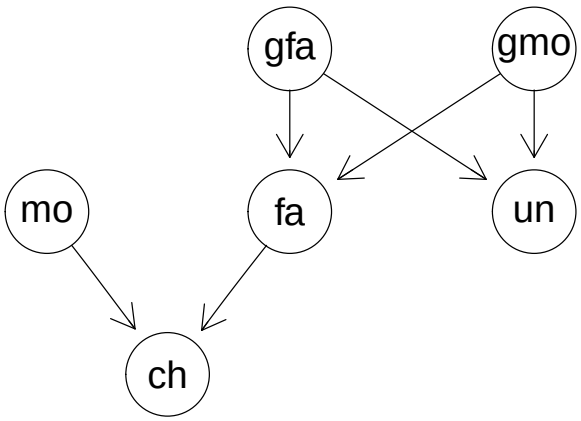
```
## P(m = yy, c = yg, f = yg)
p.fmc <- pEvidence(setEvi(trio,
                          evidence = list(mo = "yy",
                                          ch = "yg", fa = "yg")))
## P(f = yg)
p.f <- pEvidence(setEvi(trio,
```

```
                                evidence = list( fa = "yg")))
L.H1 <- p.fmc/p.f
## P(m = yy, c = yg)
L.H2 <- pEvidence(setEvi(trio,
                                 evidence = list( mo = "yy", ch = "yg")))
## Likelihood ratio comparing "Pea X" vs unknown pea.
L.H1/L.H2


## [1] 0.714
```

The likelihood ratio is smaller than 1, so the evidence does not point to Pea X being the father.

5.1 Missing father, but the uncle is available

```
dG2 <- dag(~ch|mo:fa + fa|gfa:gmo + un|gfa:gmo)
plot(dG2)
```

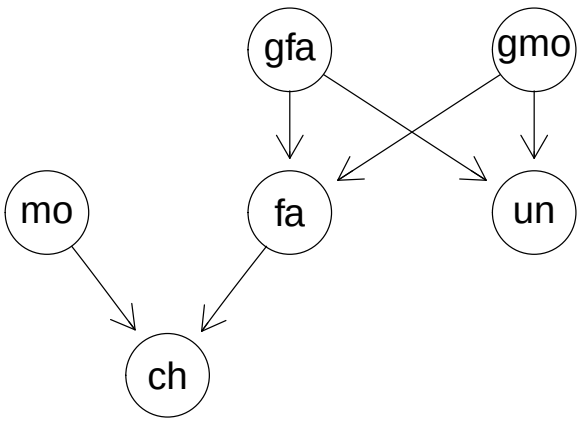


$$p(\mathtt{ch},\mathtt{mo},\mathtt{fa},\mathtt{gfa},\mathtt{gmo},\mathtt{un}) = p(\mathtt{ch}|\mathtt{mo},\mathtt{fa})p(\mathtt{fa}|\mathtt{gfa},\mathtt{gmo}) \\ p(\mathtt{un}|\mathtt{gfa},\mathtt{gmo})p(\mathtt{mo})p(\mathtt{gfa})p(\mathtt{gmo})$$

```
ssp <- list(fa = gts, mo = gts, ch = gts, ch1 = gts, ch2 = gts,
            gfa = gts, gmo = gts, un = gts)

## p(m), p(gm), p(gf)
p.m  <- ar_new(~ mo, levels=ssp, values=gtprobs2)
```

```
p.gm <- ar_new(~ gmo, levels=ssp, values=gtprobs2)
p.gf <- ar_new(~ gfa, levels=ssp, values=gtprobs2)

## p(child | mother, father)
p.c_fm <- ar_new(~ ch:mo:fa, levels=ssp, values=inheritance)

## p(father | grandma, grandpa)
p.f_gfgm <- ar_new(~ fa:gfa:gmo, levels=ssp, values=inheritance)

## p(uncle | grandma, grandpa)
p.u_gfgm <- ar_new(~ un:gfa:gmo, levels=ssp, values=inheritance)

c.mf <- parray( c("child", "mother", "father"),
levels = rep(list(gts), 3),
values = inheritance)

cpt.list <- compileCPT(list(p.c_fm, p.m, p.f_gfgm,
                            p.u_gfgm, p.gm, p.gf))
extended.family <- grain(cpt.list)
```

```
extended.family


## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:6] "ch" "mo" "fa" "un" "gmo" "gfa"


plot(extended.family)
```



5.2 Exercises

1. Build the network `extended.family` on your own computer.

2. A mother pea claims that Pea X is the father of her child pea. Unfortunately it is not possible to get a DNA sample from Pea X, but his brother ("uncle") is willing to give a sample.

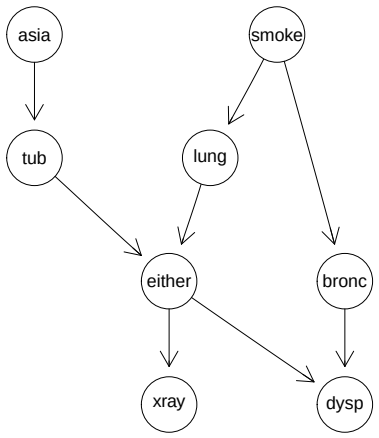
mother	yg
child	yg
uncle	gg

   What is the probability of observing this evidence, i.e. this combination of genotypes?

3. What is the conditional distribution of the father's genotype given the evidence?

4. Ignoring the genotypes of the mother and the uncle, what is the conditional distribution of the father's genotype given that the child is yg?

6 Example: The chest clinic narrative



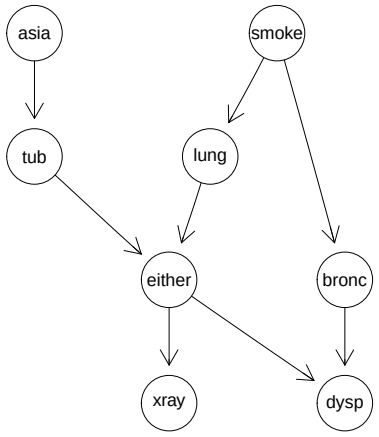
Lauritzen and Spiegehalter (1988) present the following narrative:

- "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 tuberculosis, as n*either* does the presence or absence of dyspnoea."

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

6.1 DAG–based models



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

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

6.2 Conditional probability tables (CPTs)

In R, CPTs are just multiway arrays WITH dimnames attribute. For example $p(t|a)$:

```
yn <- c("yes", "no");
x <- c(5, 95, 1, 99)
# Vanilla R
t.a <- array(x, dim=c(2, 2), dimnames=list(tub=yn, asia=yn))
t.a
```

```
##      asia
## tub   yes no
##   yes   5  1
##   no   95 99
```

```
# Alternative specification: ar_new() from gRbase
uni <- list(asia=yn, tub=yn)
t.a <- ar_new(~ tub:asia, levels=uni, values=x)
t.a
```

```
##      asia
## tub   yes no
##   yes   5  1
##   no   95 99
```

```
# Alternative (partial) specification
t.a  <- cptable(~tub | asia, values=c(5, 95, 1, 99), levels=yn)
t.a
```

```
## {v,pa(v)} : chr [1:2] "tub" "asia"
##     <NA> <NA>
## yes    5    1
## no    95   99
```

Last case: Only names of v and $pa(v)$ and levels of v are definite; the rest is inferred in the context; see later.

7 An introduction to the gRain package

7.1 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    <- 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)
```

```
cpt.list <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
cpt.list


## CPTspec 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 )
```

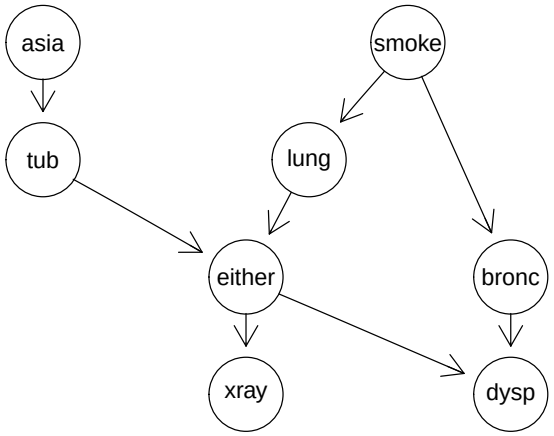
```
cpt.list$asia


## 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
```

```
# 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" ...
```

7.2 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: FALSE Propagated: FALSE
##   Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
```

The CPTs are estimated as the relative frequencies.

7.3 Querying the network

```
# Query network to find marginal probabilities of diseases
disease <- c("tub","lung","bronc")
qgrain(bn, nodes=disease)
```

```
## $tub
## tub
##    yes     no
## 0.0104 0.9896
##
## $lung
## lung
##   yes    no
## 0.055 0.945
##
## $bronc
## bronc
##  yes   no
## 0.45 0.55
```

7.4 Setting evidence

```
# Set evidence and query network again
bn.ev <- setEvi(bn, evidence=list(asia="yes",dysp="yes"))
qgrain(bn.ev, nodes=disease)
```

```
## $tub
## tub
##    yes     no
## 0.0878 0.9122
##
## $lung
```

```
## lung
##    yes     no
## 0.0995 0.9005
##
## $bronc
## bronc
##   yes    no
## 0.811 0.189
```

```
# 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
```

```
# Set additional evidence and query again
bn.ev <- setEvi(bn.ev, evidence=list(xray="yes"))
qgrain(bn.ev, nodes=disease)
```

```
## $tub
## tub
##   yes    no
## 0.392 0.608
##
## $lung
## lung
##   yes    no
## 0.444 0.556
##
## $bronc
## bronc
##   yes    no
## 0.629 0.371
```

```
# Get joint dist of tub, lung, bronc given evidence
x <- qgrain(bn.ev, nodes=disease, type="joint")
ftable( x, row.vars=1 )
```

```
##     lung      yes              no
##     bronc     yes      no      yes      no
## tub
## yes       0.01406 0.00816 0.18676 0.18274
## no        0.26708 0.15497 0.16092 0.02531
```

```
bn.ev <- retractEvidence(bn.ev, nodes="asia")
bn.ev
```

```
## Independence network: Compiled: TRUE Propagated: TRUE
##   Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
##   Evidence:
##   nodes is.hard.evidence hard.state
## 1  dysp             TRUE        yes
## 2  xray             TRUE        yes
##   pEvidence: 0.070670
```

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
## 0.0878 0.9122
##
## $lung
## lung
##    yes     no
## 0.0995 0.9005
##
## $bronc
## bronc
##   yes    no
## 0.811 0.189
```

8 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

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

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.

8.1 So what is the problem?

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?

8.2 So what is the solutions?

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.

9 Example: Lizard data

Characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H).

```
data(lizardRAW, package="gRbase")
head(lizardRAW, 4)
```

```
##   diam height species
## 1   >4  >4.75    dist
## 2   >4  >4.75    dist
## 3  <=4 <=4.75   anoli
## 4   >4 <=4.75   anoli
```

Defines 3–way contingency table.

```
data(lizard, package="gRbase")
ftable( lizard, row.vars=1 )
```

```
##      height  >4.75      <=4.75
##      species anoli dist  anoli dist
## diam
## <=4             32   61     86   73
## >4              11   41     35   70
```

9.1 Conditional independence and data

Conditional independependence $height \perp\!\!\!\perp diam | species$ (short: $h \perp\!\!\!\perp d \,|\, s$) means independence between `height` and `diam` in each `species`–slice

```
lizard
```

```
## , , species = anoli
##
```

```
##      height
## diam  >4.75 <=4.75
##   <=4    32     86
##   >4     11     35
##
## , , species = dist
##
##      height
## diam  >4.75 <=4.75
##   <=4    61     73
##   >4     41     70
```

Seems reasonable!

Tests for independence in slices support this:

```
chisq.test( lizard[, , 1] )
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  lizard[, , 1]
## X-squared = 0.05, df = 1, p-value = 0.8
```

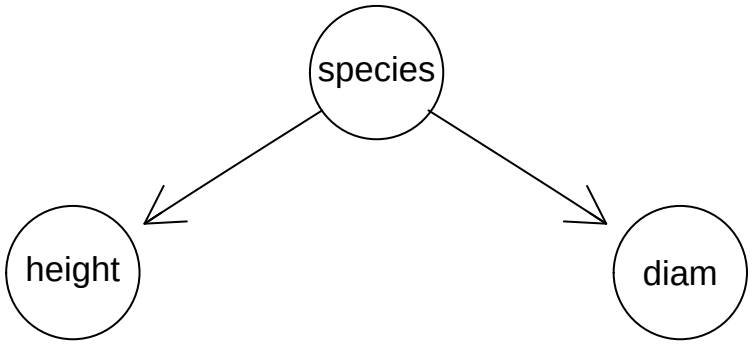
```
chisq.test( lizard[, , 2] )
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  lizard[, , 2]
## X-squared = 2, df = 1, p-value = 0.2
```

9.2 DAG factorization

A DAG that encodes $d \perp\!\!\!\perp h \,|\, s$ is

```
d <- dag( ~species + height|species + diam|species ); plot(d)
```



Joint distribution

$$p(d, h, s) = p(h|s)p(d|s)p(s)$$

The general picture: A missing edge implies a (conditional) independence: $p(d, h, s) = q_1(d, s)q_2(h, s)$.

More generally: Factorization according to DAG:

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

9.3 Extracting CPTs

```
## Extract empirical distributions
s   <- ar_marg(lizard, ~species); s

## species
## anoli  dist
##   164   245

h_s <- ar_marg(lizard, ~height + species); h_s

##         species
## height   anoli dist
##   >4.75     43  102
##   <=4.75   121  143

d_s <- ar_marg(lizard, ~diam + species); d_s

##      species
## diam  anoli dist
##   <=4   118  134
##   >4     46  111
```

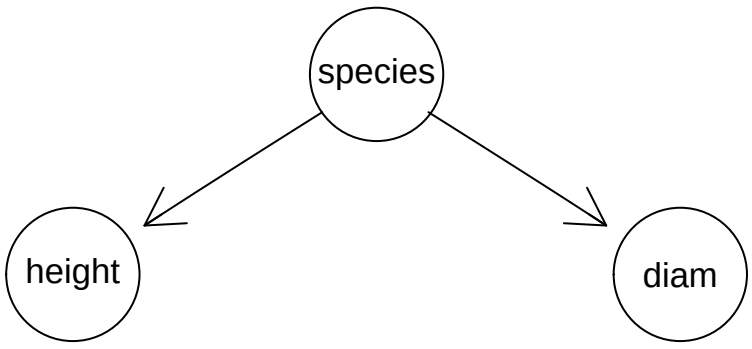
```
## Normalize to CPTs if desired (not necessary because
## we can always normalize at the end)
p.s   <- ar_normalize(s, "first"); p.s

## species
## anoli  dist
## 0.401 0.599

p.h_s <- ar_normalize(h_s, "first"); p.h_s

##         species
## height   anoli  dist
##   >4.75  0.262 0.416
##   <=4.75 0.738 0.584

p.d_s <- ar_normalize(d_s, "first"); p.d_s

##      species
## diam  anoli  dist
##   <=4  0.72 0.547
##   >4   0.28 0.453
```

We can multiply, marginalize and condition as we did before.

10 Behind the scenes: Message passing

10.1 Message passing in the lizard example

```
d <- dag( ~species + height|species + diam|species ); plot(d)
```



Joint distribution has the form

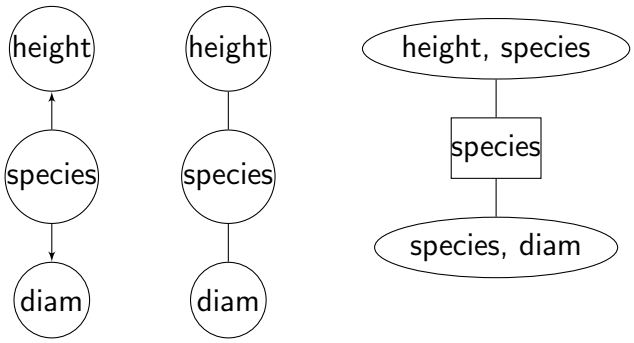
$$p(d, h, s) = p(h|s)p(d|s)p(s)$$

Terms on right hand side are given, and we can – in principle – multiply these to produce the joint distribution. (In practice we can do so in this low–dimensional case (a 2^3 table).)

However, we want to avoid forming the joint distribution and still be able to compute e.g. $p(h)$, $p(h|d)$ or $p(h|d, s)$.

From now on we no longer need the DAG. Instead we use an undirected graph to dictate the message passing:

The "moral graph" is obtained by 1) marrying parents and 2) dropping directions. The moral graph is (in this case) triangulated which means that the cliques can be organized in a tree called a junction tree.



Define $q_1(d, s) = p(s)p(d|s)$ and $q_2(h, s) = p(h|s)$ and we have

$$p(d, h, s) = p(s)p(d|s)p(h|s) = q_1(d, s)q_2(h, s)$$

The q–functions are defined on the cliques of the moral graph or - equivalently - on the nodes of the junction tree.

The q–functions are called potentials; they are non–negative functions but they are typically not probabilities and they are hence difficult to interpret. Think of q–functions as interactions.

The factorization

$$p(d, h, s) = q_1(d, s)q_2(h, s)$$

is called a *clique potential representation*.

Goal: We shall operate on q–functions such that at the end they will contain the marginal distributions, i.e.

$$q_1(d, s) = p(d, s), \quad q_2(h, s) = p(h, s)$$

10.2 Collect Evidence (II)

Pick any node, say $(height, species) = (h, s)$ as root in the junction tree, and work inwards towards the root as follows.

First, define $q_1^*(s) \leftarrow \sum_d q_1(d, s)$. We have

$$p(d, h, s) = q_1(d, s)q_2(h, s) = \left[\frac{q_1(d, s)}{q_1^*(s)}\right]\left[q_2(h, s)q_1^*(s)\right]$$

Notice that

$$p(h, s) = q_2(h, s)\sum_d q_1(d, s) = q_2(h, s)q_1^*(s) \tag{1}$$

$$p(d|h, s) = \frac{p(d, h, s)}{p(h, s)} = \frac{q_1(d, s)q_2(h, s)}{q_2(h, s)q_1^*(s)} = \frac{q_1(d, s)}{q_1^*(s)} = p(d|s) \tag{2}$$

So what we have is really

$$p(d, h, s) = \left[\frac{q_1(d, s)}{q_1^*(s)}\right]\left[q_2(h, s)q_1^*(s)\right] = p(d|s)p(h, s)$$

Therefore, we update potentials as

$$q_1(d, s) \leftarrow q_1(d, s)/q_1^*(s) = p(d|s) \quad q_2(h, s) \leftarrow q_2(h, s)q_1^*(s) = p(h, s)$$

The new potentials are also defined on the nodes of the junction tree. We still have

$$p(d, h, s) = q_1(d, s)q_2(h, s)$$

10.3 Distribute Evidence (II)

Recall that $q_2(h, s) = p(h, s)$ so we already have the clique marginal distribution on (h, s); just need $p(d, s)$.

Next work outwards from the root (h, s).

Set $q_2^*(s) \leftarrow \sum_h q_2(h,s) = \sum_h p(h,s) = p(s)$. So now

$$p(d,s) = p(d|s)p(s) = q_1(d,s)q_2^*(s).$$

Hence:

$$p(d,h,s) = q_1(d,s)q_2(h,s) = \frac{\big[q_1(d,s)q_2^*(s)\big]q_2(h,s)}{q_2^*(s)} = \frac{\big[p(d,s)\big]p(h,s)}{p(s)}$$

We therefore update as $q_1(d,s) \leftarrow q_1(d,s)q_2^*(s) = p(d|s)p(s) = p(d,s)$ and have

$$p(d,h,s) = \frac{q_1(d,s)q_2(h,s)}{q_2^*(s)} = \frac{p(d,s)p(h,s)}{p(s)}$$

The form above is called the *clique marginal representation* and the main point is that q_1 and q_2 "fit on their marginals", i.e. $q_1(s) = q_2(s)$ and

$$q_1(d,s) = p(d,s), \quad q_2(h,s) = p(h,s)$$

10.4 Collect and distribute evidence - in R

Before we extracted the CPTs from data

```
list(p.s, p.d_s, p.h_s)
```

```
## [[1]]
## species
## anoli  dist
## 0.401 0.599
##
## [[2]]
##      species
## diam  anoli  dist
##   <=4  0.72 0.547
##   >4   0.28 0.453
##
## [[3]]
##         species
## height   anoli  dist
##   >4.75  0.262 0.416
##   <=4.75 0.738 0.584
```

Define q_1 and q_2:

```
q1.ds <- ar_prod(p.s, p.d_s); q1.ds

##      species
## diam  anoli  dist
##   <=4 0.289 0.328
##   >4  0.112 0.271

q2.hs <- p.h_s; q2.hs

##         species
## height   anoli  dist
##   >4.75  0.262 0.416
##   <=4.75 0.738 0.584
```

Collect evidence:

$$q_1(s) \leftarrow \sum_d q_1(s,b); q_1(d,s) \leftarrow q_1(d,s)/q_1(s); q_2(h,s) \leftarrow q_2(h,s)q_1(s)$$

is done as follows:

```
q1.s  <- ar_marg(q1.ds, "species"); q1.s

## species
## anoli  dist
## 0.401 0.599

q1.ds <- ar_div(q1.ds, q1.s); q1.ds

##        diam
## species   <=4    >4
##   anoli 0.720 0.280
##   dist  0.547 0.453

q2.hs <- ar_mult(q2.hs, q1.s); q2.hs

##        height
## species >4.75 <=4.75
##   anoli 0.105  0.296
##   dist  0.249  0.350
```

Distribute evidence:

```
q2.s  <- ar_marg(q2.hs, "species"); q2.s

## species
## anoli  dist
## 0.401 0.599

q1.ds <- ar_mult(q1.ds, q2.s); q1.ds

##         diam
## species   <=4    >4
##   anoli 0.289 0.112
##   dist  0.328 0.271
```

10.5 It works - empirical proof (II)

The joint distribution $p(d, h, s)$ is a $2 \times 2 \times 2$ array which we really do not want to calculate this in general; here we just do it as "proof of concept":

```
p.dhs <- ar_prod( p.s, p.d_s, p.h_s )
ftable( p.dhs, row.vars="species")

##         height  >4.75        <=4.75
##         diam      <=4     >4    <=4     >4
## species
## anoli          0.0756 0.0295 0.2129 0.0830
## dist           0.1364 0.1130 0.1912 0.1584
```

Claim: After these steps $q_1(d, s) = p(d, s)$ and $q_2(h, s) = p(h, s)$. That is, we have the marginal distributions on the cliques:

Proof:

```
q1.ds

##         diam
## species   <=4    >4
##   anoli 0.289 0.112
##   dist  0.328 0.271

ar_dist(p.dhs, ~ diam + species)

##      species
```

```
## diam  anoli  dist
##   <=4 0.289 0.328
##   >4  0.112 0.271
```

```
q2.hs
```

```
##         height
## species >4.75 <=4.75
##   anoli 0.105  0.296
##   dist  0.249  0.350
```

```
ar_dist(p.dhs, ~ height + species)
```

```
##         species
## height   anoli  dist
##   >4.75  0.105 0.249
##   <=4.75 0.296 0.350
```

Now we can obtain, e.g. $p(h)$ as

```
p.h <- ar_dist(q2.hs, "height")
p.h
```

```
## height
##  >4.75 <=4.75
##  0.355  0.645
```

And we NEVER calculated the full joint distribution!

10.6 Setting evidence

Next consider the case where we have the evidence that `diam>4`.

Since we have calculated the joint pmf, we can use it for calculating $p(h, s|d = >4)$:

```
p.hs_d <- ar_slice(p.dhs, slice=list(diam=">4"))
p.hs_d <- p.hs_d / sum(p.hs_d)
p.hs_d
```

```
##         species
## height    anoli  dist
##   >4.75  0.0768 0.294
##   <=4.75 0.2162 0.413
```

Alternatively, we can set all entries in $p(d, h, s)$ that are not compatible with `diam>4` equal to zero and normalize the result:

```
p2.dhs <- p.dhs
p2.dhs <- ar_slice_mult(p2.dhs, list(diam=">4"))
p2.dhs <- p2.dhs / sum(p2.dhs)
```

Clearly

```
p2.dhs / sum(p2.dhs)
```

```
## , , diam = <=4
##
##         species
## height   anoli dist
##   >4.75      0    0
##   <=4.75     0    0
##
## , , diam = >4
##
##         species
## height    anoli  dist
##   >4.75  0.0768 0.294
##   <=4.75 0.2162 0.413
```

So evidence can be intered by just modifying the distribution by setting entries not compatible with the evidence to have probability 0 and then normalize the result at the end.

10.7 Entering evidence - in R

We can do the same thing in the message passing scheme:
Define q_1 and q_2 (must do so because they were modified before):

```
q1.ds <- ar_prod(p.s, p.d_s)
q2.hs <- p.h_s
q1.ds <- ar_slice_mult(q1.ds, list(diam=">4"))
q1.ds
```

```
##      species
## diam  anoli  dist
##   <=4 0.000 0.000
##   >4  0.112 0.271
```

```
# Repeat all the same steps as before
q1.s  <- ar_marg(q1.ds, "species")
q1.ds <- ar_div(q1.ds, q1.s)
q2.hs <- ar_mult(q2.hs, q1.s)
q2.s  <- ar_marg(q2.hs, "species")
q1.ds <- ar_mult(q1.ds, q2.s)
```

Claim: After these steps $q_1(d, s) \propto p(d, s|d^+)$ and $q_2(h, s) \propto p(h, s|d^+)$.

```
p0.dhs <- ar_slice_mult(p.dhs, list(diam=">4"))
ftable( p0.dhs, row.vars=1)
```

```
##        species  anoli          dist
##        diam       <=4     >4    <=4     >4
## height
## >4.75          0.0000 0.0295 0.0000 0.1130
## <=4.75         0.0000 0.0830 0.0000 0.1584
```

Proof:

```
q1.ds / sum( q1.ds ) ## Needs normalization
```

```
##        diam
## species <=4    >4
##   anoli   0 0.293
##   dist    0 0.707
```

```
ar_dist(p0.dhs, ~ diam:species)
```

```
##      species
## diam  anoli  dist
##   <=4 0.000 0.000
##   >4  0.293 0.707
```

```
ar_dist(p.dhs, ~ species, cond=list(diam=">4"))
```

```
## species
## anoli  dist
## 0.293 0.707
```

```
q2.hs / sum( q2.hs ) ## Needs normalization
```

```
##        height
```

```
## species  >4.75 <=4.75
##   anoli 0.0768  0.216
##   dist  0.2943  0.413
```

```
ar_dist(p0.dhs, ~ height:species)
```

```
##         species
## height    anoli  dist
##   >4.75  0.0768 0.294
##   <=4.75 0.2162 0.413
```

```
ar_dist(p.dhs, ~ height:species, cond=list(diam=">4"))
```

```
##         species
## height    anoli  dist
##   >4.75  0.0768 0.294
##   <=4.75 0.2162 0.413
```

And we NEVER calculated the full joint distribution!

11 Learning – from data to BNs

- If we know the DAG, we can estimate the CPTs from data, e.g. as relative frequencies. Known as *parameter learning*.
- If we don't know the DAG then we can find a DAG from data using statistical methods. Known as *structural learning*.
- From the perspection of computations in the network, the DAG is not used. All computations are based on a junction tree.
- If we know a junction tree we can estimate clique marginals from data.
- If we do not know either, we can find a junction tree from data using statistical methods.
- One way ahead: A junction tree corresponds to a special type of statistical model: A decomposable graphical model.
- We shall use data for finding a decomposable graphical model, and then convert this to a BN.

12 Discrete data and contingency tables

Characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H).

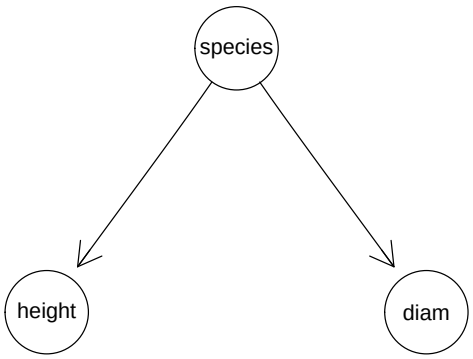
```
data(lizardRAW, package="gRbase")
head(lizardRAW, 4)


##   diam height species
## 1   >4  >4.75    dist
## 2   >4  >4.75    dist
## 3  <=4 <=4.75   anoli
## 4   >4 <=4.75   anoli
```

Defines 3–way contingency table.

```
data(lizard, package="gRbase")
ftable( lizard, row.vars=1 )


##      height  >4.75      <=4.75
##      species anoli dist  anoli dist
## diam
## <=4             32   61     86   73
## >4              11   41     35   70
```

12.1 Extracting CPTs

```
d <-dag( ~species + height|species + diam|species ); plot(d)
```



Joint distribution

$$p(s, b, d) = p(h|s)p(d|s)p(s)$$

```
## Extract empirical distributions
s   <- ar_marg(lizard, ~species);
s   <- ar_normalize(s, type="first"); s


## species
## anoli  dist
## 0.401 0.599


h_s <- ar_marg(lizard, ~height + species);
h_s <- ar_normalize(h_s, type="first"); h_s


##         species
## height   anoli  dist
##   >4.75  0.262 0.416
##   <=4.75 0.738 0.584


d_s <- ar_marg(lizard, ~diam + species);
d_s <- ar_normalize(d_s, type="first"); d_s


##      species
## diam  anoli  dist
##   <=4  0.72 0.547
##   >4   0.28 0.453
```

12.2 Creating BN from CPTs

```
cpt.list <- compileCPT(list(s, h_s, d_s)); cpt.list


## CPTspec with probabilities:
##  P( species )
##  P( height | species )
##  P( diam | species )


net <- grain( cpt.list ); net


## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:3] "species" "height" "diam"
```

12.3 Exercise - different lizard models

1. There are three DAGs for the lizard data that will encode the conditional independence restriction $D \perp\!\!\!\perp H \mid S$. Create and display these.
2. Write down the corresponding factorizations of the joint distribution.
3. Extract the corresponding CPTs from data, and create the three corresponding Bayesian networks from the CPTs.
4. What do you conclude about these networks? (Do they produce the same marginal and conditional probabilities?)
5. What does that lead you to conclude with respect to the importance of the directions of the arrows in the dag?

13 Log–linear models

Goal: We shall discuss *log–linear* models for discrete data, and we shall end up focusing on a specific class of log–linear models: The *decomposable graphical* log–linear models.

We take this focus because these models can be transformed into an equivalent *Bayesian network*.

13.1 Data formats

The `lizard` data comes in various forms:

A raw case list A dataframe in which each row represents an observation

```
data(lizardRAW, package="gRbase")
dim(lizardRAW)

## [1] 409   3

head(lizardRAW)
```

```
##   diam height species
## 1   >4  >4.75    dist
## 2   >4  >4.75    dist
## 3  <=4 <=4.75   anoli
## 4   >4 <=4.75   anoli
## 5   >4 <=4.75    dist
## 6  <=4 <=4.75   anoli
```

Each row represents a vector $x = (x_D, x_H, x_S) = (d, h, s)$ where

$$\begin{aligned} d \in \{>4, \leq 4\} &\equiv \{d^+, d^-\} = \mathcal{D} \\ h \in \{>4.75; \leq 4.75\} &\equiv \{h^+, h^-\} = \mathcal{H} \\ s \in \{\text{dist; anoli}\} &\equiv \{s^+, s^-\} = \mathcal{S} \end{aligned}$$

So $x = (d, h, s)$ takes values in $\mathcal{X} = \mathcal{D} \times \mathcal{H} \times \mathcal{S}$.

A contingency table If we count the number of occurences $n(d, h, s)$ of each configuration (d, h, s) of the variables we can organize data as a contingency table

$$n = \Big(n(x); x \in \mathcal{X}\Big) = \Big(n(d, h, s); (d, h, s) \in \mathcal{D} \times \mathcal{H} \times \mathcal{S}\Big)$$

```
lizard <- xtabs( ~ diam + height + species, data=lizardRAW)
lizard
```

```
## , , species = anoli
##
##      height
## diam  <=4.75 >4.75
##   <=4     86    32
##   >4      35    11
##
## , , species = dist
##
##      height
## diam  <=4.75 >4.75
##   <=4     73    61
##   >4      70    41
```

which is a $2 \times 2 \times 2$ array. Same as

```
data(lizard, package="gRbase")
```

Aggregated case list These frequencies can also be represented as a dataframe

```
lizardAGG <- as.data.frame(lizard)
head(lizardAGG)
```

```
##   diam height species Freq
## 1  <=4  >4.75   anoli   32
## 2   >4  >4.75   anoli   11
## 3  <=4 <=4.75   anoli   86
## 4   >4 <=4.75   anoli   35
## 5  <=4  >4.75    dist   61
## 6   >4  >4.75    dist   41
```

Same as

```
data(lizardAGG, package="gRbase")
```

13.2 Modelling discrete data

```
head(lizardRAW, 4)
```

```
##   diam height species
## 1   >4  >4.75    dist
## 2   >4  >4.75    dist
## 3  <=4 <=4.75   anoli
## 4   >4 <=4.75   anoli
```

Above, each row x^i can be seen as a realization of the random vector $X^i = (X_D^i, X_H^i, X_S^i)$.

We write a generic observation as $x = (x_D, x_H, x_S) = (d, h, s)$, where

$$
\begin{aligned}
d \in \{> 4, \leq 4\} &\equiv \{d^+, d^-\} = \mathcal{D} \\
h \in \{> 4.75; \leq 4.75\} &\equiv \{h^+, h^-\} = \mathcal{H} \\
s \in \{\text{dist}; \text{anoli}\} &\equiv \{s^+, s^-\} = \mathcal{S}
\end{aligned}
$$

The number of observations (rows) with the configuration $x = (d, h, s)$ is denoted $n(x)$ or $n(d, h, s)$.

The joint probability of observing data as a case list is

$$p(x^i; i = 1, \ldots N) = \prod_{i=1}^{N} p(x^i) = \prod_{x \in \mathcal{X}} p(x)^{n(x)}$$

where $Pr(X = x) = p(x)$ for $x \in \mathcal{X}$.

The joint probability of the corresponding observed contingency table is

$$p(\{n(x); x \in \mathcal{X}\}) = \frac{N!}{\prod_{x \in \mathcal{X}} n(x)!} \prod_{x \in \mathcal{X}} p(x)^{n(x)}$$

If we let $p = (p(x); x \in \mathcal{X})$ the likelihood is in both cases

$$L(p) \propto \prod_{x \in \mathcal{X}} p(x)^{n(x)}$$

Alternatively, we can assume that counts $n(x)$ are independent realizations of Poisson variates $N(x)$ with mean $m(x)$. In that case, the likelihood for $m = (m(x); x \in \mathcal{X})$ is

$$L(m) \propto \prod_{x \in \mathcal{X}} m(x)^{n(x)}$$

This Poisson likelihood is proportional to the binomial likelihood since under binomial sampling, the expected counts in cell x is $m(x) = Np(x)$. The MLE for $\sum_{x \in \mathcal{X}} m(x)$ is $\sum_{x \in \mathcal{X}} n(x) = N$.

This is called the *Poisson trick* and allows log–linear models to be fitted with `glm()`.

If we put no restriction on $p(x)$ (other than $p(x) \geq 0$, and $\sum_{x \in \mathcal{X}} p(x) = 1$), the MLE for p is $\hat{p}(x) = n(x)/N$ This is called the *saturated model*. So for the saturated model, the MLE for the expectation is $\hat{m}(x) = n(x)$.

To be more specific, we return to the `lizard` data:

```
head(lizardRAW, 4)
```

```
##   diam height species
## 1   >4  >4.75    dist
## 2   >4  >4.75    dist
## 3  <=4 <=4.75   anoli
## 4   >4 <=4.75   anoli
```

The probability of data as a case list is

$$p(x^i, i = 1, \ldots, N) = \prod_{i=1}^{N} p(x_D^i, x_H^i, x_S^i) = \prod_{i=1}^{N} p(d^i, h^i, s^i) = \prod_{d,h,s} p_{dhs}^{n_{dhs}}$$

where n_{dhs} is the number of observations (rows) with the configuration (d, h, s).

The joint probability of the observed contingency table is

$$p((n(d, h, s); (d, h, s) \in \mathcal{X})) = \frac{n!}{\prod_{dhs} n(d, h, s)!} \prod_{d,h,s} p_{dhs}^{n_{dhs}}$$

When it comes to the likelihood we can therefore think of data as a case list or as a contingency table as we like; the likelihood is

$$L(p) \propto \prod_{d,h,s} p(d, h, s)^{n(d,h,s)}$$

For the *Poisson trick* we have that the expected counts in cell (d, h, s) is $m(d, h, s) = Np(d, h, s)$, and

$$L(p) \propto \prod_{d,h,s} m(d, h, s)^{n(d,h,s)}; \quad m(d, h, s) = Np(d, h, s)$$

13.3 Hierarchical log–linear models

The saturated model for $p(d, h, s)$ is usually not too interesting; instead we restrict $p(d, h, s)$ to obtain a more parsimoneous (economical) description and/or to exploit structural information.

We model the *cell probabilities* (or *expected cell counts*) by an ANOVA–like factorial expansion into interaction terms:

$$\log p(d, h, s) = \alpha^0 + \alpha_d^D + \alpha_h^H + \alpha_s^S + \beta_{dh}^{DH} + \beta_{ds}^{DS} + \beta_{hs}^{HS} + \gamma_{dhs}^{DHS}$$

Because of this log–linear expansions, the models are called *log–linear models*.

Think of d taking values d^+ or d^- and similarly for h and s.

Above, β_{dh}^{DH} really refers to four parameters values:

$$\beta_{d^+h^+}^{DH}, \quad \beta_{d^-h^+}^{DH}, \quad \beta_{d^+h^-}^{DH}, \quad \beta_{d^-h^-}^{DH}.$$

As such, the expression above is greatly *over-parameterized* because there are $1+2+2+2+4+4+4+8=27$ parameters but only 8 cells in the contingency table.

Parameters can be made *identifiable* by e.g. the restriction that "whenever there is a plus in the subscript" the parameter is set to zero. Hence $\beta^{DH}_{d^+h^+} = \beta^{DH}_{d^-h^+} = \beta^{DH}_{d^+h^-} = 0$.

With that convention, there are only 8 parameters above.

13.4 A model which implies conditional independence

Structure on the model is obtained by setting terms to zero.

For example, we set $\beta^{DH}_{dh} = \gamma^{DHS}_{dhs} = 0$:

$$\log p_{dhs} = \alpha^0 + \alpha^D_d + \alpha^H_h + \alpha^S_s + \beta^{DS}_{ds} + \beta^{HS}_{hs}$$

The *generators* of the model is just a short way of writing the terms in the model:

`[D] [H] [S] [DS] [HS]`

Terms that are contained in other terms can be ignored. For example the *main effect* α^D_d is contained in the *two–way interaction* β^{DS}_{ds} so α^D_d can be ignored. Likewise α^0 can be ignored and we write

$$\log p_{dhs} = \tilde{\beta}^{DS}_{ds} + \tilde{\beta}^{HS}_{hs} \tag{3}$$

Generators:

`[DS] [HS]`

Exponentiating in (3) gives

$$p_{dhs} = \exp(\tilde{\beta}^{DS}_{ds}) \exp(\tilde{\beta}^{HS}_{hs}) = q_1(d,s) q_2(h,s)$$

Hence the *factorization criterion* gives directly that $D \perp\!\!\!\perp H \mid S$.

We shall illustrate in terms of a model matrix

```
lizardAGG2 <- lizardAGG
names(lizardAGG2)[1:3] <- c("D", "H", "S")
#levels(lizardAGG2£D) = c("d+", "d-")
#levels(lizardAGG2£H) = c("h+", "h-")
#levels(lizardAGG2£S) = c("s+", "s-")
lizardAGG2
```

```
##     D      H     S Freq
## 1 <=4  >4.75 anoli   32
## 2  >4  >4.75 anoli   11
## 3 <=4 <=4.75 anoli   86
## 4  >4 <=4.75 anoli   35
## 5 <=4  >4.75  dist   61
## 6  >4  >4.75  dist   41
## 7 <=4 <=4.75  dist   73
## 8  >4 <=4.75  dist   70
```

In R terminology we write

```
X1 <- model.matrix(~ D + H + S + D : S + H : S, data=lizardAGG2)
X1 %>% prmatrix
```

```
##   (Intercept) D>4 H>4.75 Sdist D>4:Sdist H>4.75:Sdist
## 1           1   0      1     0         0            0
## 2           1   1      1     0         0            0
## 3           1   0      0     0         0            0
## 4           1   1      0     0         0            0
## 5           1   0      1     1         0            1
## 6           1   1      1     1         1            1
## 7           1   0      0     1         0            0
## 8           1   1      0     1         1            0
```

In R terminology we write

```
X2 <- model.matrix(~ 0 + D : S + H : S, data=lizardAGG2)
X2 %>% prmatrix
```

```
##   D<=4:Sanoli D>4:Sanoli D<=4:Sdist D>4:Sdist Sanoli:H>4.75
## 1           1          0          0         0             1
## 2           0          1          0         0             1
## 3           1          0          0         0             0
## 4           0          1          0         0             0
## 5           0          0          1         0             0
## 6           0          0          0         1             0
## 7           0          0          1         0             0
## 8           0          0          0         1             0
```

```
##   Sdist:H>4.75
## 1            0
## 2            0
## 3            0
## 4            0
## 5            1
## 6            1
## 7            0
## 8            0
```

The two model matrices must define the same model

```
library(lme4)
c(rankMatrix(X1), rankMatrix(X2), rankMatrix(cbind(X1, X2)))


## [1] 6 6 6
```

13.5 Some other models

Other models frequently encountered:

saturated model: No terms are set to zero

$$\log p_{dhs} = \alpha^0 + \alpha_d^D + \alpha_h^H + \alpha_s^S + \beta_{dh}^{DH} + \beta_{ds}^{DS} + \beta_{hs}^{HS} + \gamma_{dhs}^{DHS}$$

Generators:

`[D] [H] [S] [DH] [DS] [HS] [DHS] (same as) [DHS]`

all two-way interaction model: Only the three-way interaction is set to zero.

$$\log p_{dhs} = \alpha^0 + \alpha_d^D + \alpha_h^H + \alpha_s^S + \beta_{dh}^{DH} + \beta_{ds}^{DS} + \beta_{hs}^{HS}$$

Generators:

`[D] [H] [S] [DH] [DS] [HS] (same as) [DH] [DS] [HS]`

independence model: All interaction terms are set to zero

$$\log p_{dhs} = \alpha^0 + \alpha_d^D + \alpha_h^H + \alpha_s^S$$

Generators:

`[D] [H] [S]`

13.6 How to - in R

Log–linear models can be fitted with `glm()` due to the *Poisson trick*:

```
library(broom)
head(lizardAGG, 4)

##   diam height species Freq
## 1  <=4  >4.75   anoli   32
## 2   >4  >4.75   anoli   11
## 3  <=4 <=4.75   anoli   86
## 4   >4 <=4.75   anoli   35

lizardAGG2 <- lizardAGG
names(lizardAGG2)[1:3] <- c("D", "H", "S")
m1.ci <- glm(Freq ~ 0 + D:S + H:S, family=poisson, data=lizardAGG2)
tidy(m1.ci)

##            term estimate std.error statistic   p.value
## 1   D<=4:Sanoli    4.467     0.103     43.30  0.00e+00
## 2    D>4:Sanoli    3.525     0.155     22.80 5.06e-115
## 3    D<=4:Sdist    4.359     0.102     42.80  0.00e+00
## 4     D>4:Sdist    4.171     0.109     38.20  0.00e+00
## 5 Sanoli:H>4.75   -1.035     0.178     -5.83  5.63e-09
## 6  Sdist:H>4.75   -0.338     0.130     -2.61  9.13e-03

m1.sat <- glm(Freq ~ 0 + D:H:S, family=poisson, data=lizardAGG2)
tidy(m1.sat)

##                  term estimate std.error statistic   p.value
## 1 D<=4:H<=4.75:Sanoli     4.45     0.108     41.31  0.00e+00
## 2  D>4:H<=4.75:Sanoli     3.56     0.169     21.03  3.22e-98
## 3  D<=4:H>4.75:Sanoli     3.47     0.177     19.61  1.40e-85
## 4   D>4:H>4.75:Sanoli     2.40     0.302      7.95  1.82e-15
## 5  D<=4:H<=4.75:Sdist     4.29     0.117     36.66 3.45e-294
## 6   D>4:H<=4.75:Sdist     4.25     0.120     35.55 9.76e-277
## 7   D<=4:H>4.75:Sdist     4.11     0.128     32.11 3.53e-226
## 8    D>4:H>4.75:Sdist     3.71     0.156     23.78 5.58e-125

m1.ind <- glm(Freq ~ 0 + D + H + S, family=poisson, data=lizardAGG2)
tidy(m1.ind)

##     term estimate std.error statistic   p.value
## 1   D<=4    4.178    0.0947     44.13  0.00e+00
## 2    D>4    3.705    0.1066     34.75 1.36e-264
## 3 H>4.75   -0.599    0.1034     -5.80  6.75e-09
## 4  Sdist    0.401    0.1009      3.98  6.94e-05
```

Model comparisons

```
anova(m1.sat, m1.ci, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: Freq ~ 0 + D:H:S
## Model 2: Freq ~ 0 + D:S + H:S
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         0       0.00
## 2         2       2.03 -2    -2.03     0.36
```

```
anova(m1.sat, m1.ind, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: Freq ~ 0 + D:H:S
## Model 2: Freq ~ 0 + D + H + S
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         0          0
## 2         4         25 -4      -25  4.9e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(m1.ci, m1.ind, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: Freq ~ 0 + D:S + H:S
## Model 2: Freq ~ 0 + D + H + S
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         2       2.03
## 2         4      25.04 -2      -23    1e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
AIC(m1.sat, m1.ci, m1.ind)
```

```
##        df AIC
## m1.sat  8  61
## m1.ci   6  59
## m1.ind  4  78
```

Log-linear models can also be fitted with `loglm()` which takes data in the form of a contingency table:

```r
lizard2 <- lizard
names(dimnames(lizard2)) <- c("D", "H", "S")
lizard2
```

```
## , , S = anoli
##
##      H
## D     >4.75 <=4.75
##   <=4    32     86
##   >4     11     35
##
## , , S = dist
##
##      H
## D     >4.75 <=4.75
##   <=4    61     73
##   >4     41     70
```

```r
library(MASS) ## to get loglm
m2.ci <- loglm( ~ D:S + H:S, data=lizard2)
m2.ci
```

```
## Call:
## loglm(formula = ~D:S + H:S, data = lizard2)
##
## Statistics:
##                      X^2 df P(> X^2)
## Likelihood Ratio 2.03  2    0.363
## Pearson          2.02  2    0.365
```

```r
m2.sat <- loglm( ~ D:H:S, data=lizard2)
m2.sat
```

```
## Call:
## loglm(formula = ~D:H:S, data = lizard2)
##
## Statistics:
##                     X^2 df P(> X^2)
## Likelihood Ratio   0  0        1
## Pearson            0  0        1
```

```r
m2.ind <- loglm( ~ D + H + S, data=lizard2)
m2.ind
```

```
## Call:
## loglm(formula = ~D + H + S, data = lizard2)
```

```
##
## Statistics:
##                   X^2 df P(> X^2)
## Likelihood Ratio 25.0  4 4.95e-05
## Pearson          23.9  4 8.34e-05
```

```
anova(m2.sat, m2.ci, test="Chisq")
```

```
## LR tests for hierarchical log-linear models
##
## Model 1:
##  ~D:S + H:S
## Model 2:
##  ~D:H:S
##
##           Deviance df Delta(Dev) Delta(df) P(> Delta(Dev)
## Model 1       2.03  2
## Model 2       0.00  0       2.03         2          0.363
## Saturated     0.00  0       0.00         0          1.000
```

```
anova(m2.sat, m2.ind, test="Chisq")
```

```
## LR tests for hierarchical log-linear models
##
## Model 1:
##  ~D + H + S
## Model 2:
##  ~D:H:S
##
##           Deviance df Delta(Dev) Delta(df) P(> Delta(Dev)
## Model 1         25  4
## Model 2          0  0         25         4          5e-05
## Saturated        0  0          0         0          1e+00
```

```
anova(m2.ci, m2.ind, test="Chisq")
```

```
## LR tests for hierarchical log-linear models
##
## Model 1:
##  ~D + H + S
## Model 2:
##  ~D:S + H:S
##
##           Deviance df Delta(Dev) Delta(df) P(> Delta(Dev)
## Model 1      25.04  4
## Model 2       2.03  2      23.01         2        0.00001
## Saturated     0.00  0       2.03         2        0.36319
```

Alas, we can not get AIC for these models directly (someone should fix that)

```
## AIC(m2.sat, m2.ci, m2.ind)
```

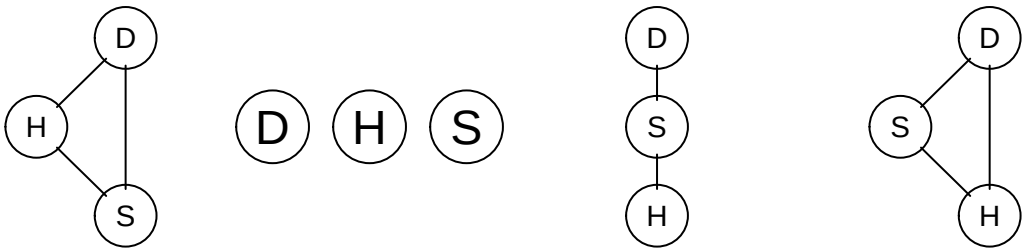
13.7 Differences

The *IPS* algorithm behind `loglm()` is generally better than the *iteratively reweigthed least squares* algorithm behind `glm()`.

13.8 Dependence graphs

A *dependence graph* for a log-linear model is given as follows: Variables are nodes; Edges are given by this rule: Whenever two variables are in the same generator there must be an edge between them:

```
par(mfrow=c(1,4))
at <- list(node=list(fontsize=5))
plot( (g1 <- ug(~D:H:S) ), attrs=at)
plot( (g2 <- ug(~D + H + S) ), attrs=at)
plot( (g3 <- ug(~D:S + H:S) ), attrs=at)
plot( (g4 <- ug(~D:S + H:S + D:H) ), attrs=at)
```



The *cliques* of the dependence graph of a model are sometimes the same as the generators of a model:

```
str( getCliques( g1 ) )

## List of 1
##  $ : chr [1:3] "D" "H" "S"

str( getCliques( g2 ) )
```

```
## List of 3
##  $ : chr "D"
##  $ : chr "H"
##  $ : chr "S"


str( getCliques( g3 ) )


## List of 2
##  $ : chr [1:2] "S" "D"
##  $ : chr [1:2] "S" "H"


str( getCliques( g4 ) ) ## No - not here!


## List of 1
##  $ : chr [1:3] "D" "S" "H"
```

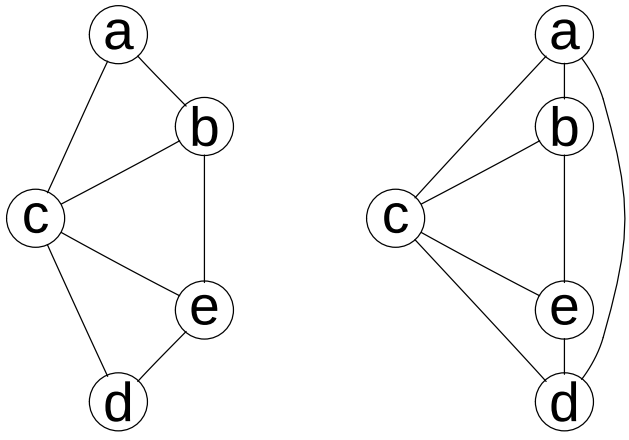
When the generators and the cliques of the dependence graph are the same, the model is said to be a *graphical* log–linear model.

13.9 Decomposable log–linear models

Decomposable log–linear models: graphical models whose dependence graph is *triangulated*

If there are no 4–cycles the graph is triangulated.

```
par( mfrow=c(1,2))
plot( ug(~a:b:c + b:c:e + c:d:e) ) # triangulated
plot( ug(~a:b:c + b:c:e + c:d:e + a:c:d) ) # not triangulated
```



13.10 Testing for conditional independence

Tests of general conditional independence $u \perp\!\!\!\perp v \,|\, W$ can be performed with *ciTest()* (a wrapper for calling *ciTest_table()*).

The general syntax of the set argument is of the form (u, v, W) where u and v are variables and W is a set of variables.

```
ciTest(lizard, set=c("diam", "height", "species"))


## Testing diam _|_ height | species
## Statistic (DEV):    2.026 df: 2 p-value: 0.3632 method: CHISQ
```

Alternative forms are available:

```
ciTest(lizard, set= ~ diam + height + species)
ciTest(lizard, ~ di + he + s)
ciTest(lizard, c("di", "he", "sp"))
ciTest(lizard, c(2, 3, 1))
```

13.11 What is a CI-test – stratification

Conditional independence $u \perp\!\!\!\perp v \,|\, W$ means independence of u and v for each configuration w^* of W.

Conceptually form a factor S by crossing the factors in W. The test is then a test of the conditional independence $u \perp\!\!\!\perp v \,|\, S$ in a three way table.

The deviance decomposes into independent contributions from each stratum:

$$\begin{aligned} D &= 2\sum_{ijs} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} \\ &= \sum_{s} 2\sum_{ij} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_{s} D_s \end{aligned}$$

where the contribution D_s from the sth slice is the deviance for the independence model of u and v in that slice.

The test by *ciTest()* corresponds to the test for removing the edge $\{u, v\}$ from the saturated model with variables $\{u, v\} \cup W$.

```
lizard
```

```
## , , species = anoli
##
##      height
## diam  >4.75 <=4.75
##   <=4    32     86
##   >4     11     35
##
## , , species = dist
##
##      height
## diam  >4.75 <=4.75
##   <=4    61     73
##   >4     41     70
```

```
cit <- ciTest(lizard, set= ~ diam + height + species, slice.info=T)
cit
```

```
## Testing diam _|_ height | species
## Statistic (DEV):    2.026 df: 2 p-value: 0.3632 method: CHISQ
```

```
names(cit)
```

```
## [1] "statistic" "p.value"   "df"        "statname"  "method"
## [6] "adjust.df" "varNames"  "slice"
```

```
cit$slice
```

```
##   statistic p.value df species
## 1     0.178   0.673  1   anoli
## 2     1.848   0.174  1    dist
```

The sth slice is a $|u| \times |v|$–table $\{n_{ijs}\}_{i=1...|u|,j=1...|v|}$. The degrees of freedom corresponding to the test for independence in this slice is

$$df_s = (\#\{i : n_{i\cdot s} > 0\} - 1)(\#\{j : n_{\cdot js} > 0\} - 1)$$

where $n_{i\cdot s}$ and $n_{\cdot js}$ are the marginal totals.

13.12 Exercise: UCBA admissions

```r
ftable(UCBAdmissions, row.vars=c("Dept","Admit"))
```

```
##               Gender Male Female
## Dept Admit
## A    Admitted         512     89
##      Rejected         313     19
## B    Admitted         353     17
##      Rejected         207      8
## C    Admitted         120    202
##      Rejected         205    391
## D    Admitted         138    131
##      Rejected         279    244
## E    Admitted          53     94
##      Rejected         138    299
## F    Admitted          22     24
##      Rejected         351    317
```

1. Read the documentation about the `UCBAdmissions` data carefully.
2. Test the hypothesis that `Admit` $\perp\!\!\!\perp$ `Gender`.
3. Test the hypothesis that `Admit` $\perp\!\!\!\perp$ `Gender` | `Dept`.
4. Use the `slice.info` feature of `ciTest()` to investigate the reason for the conclusion in the previous question.

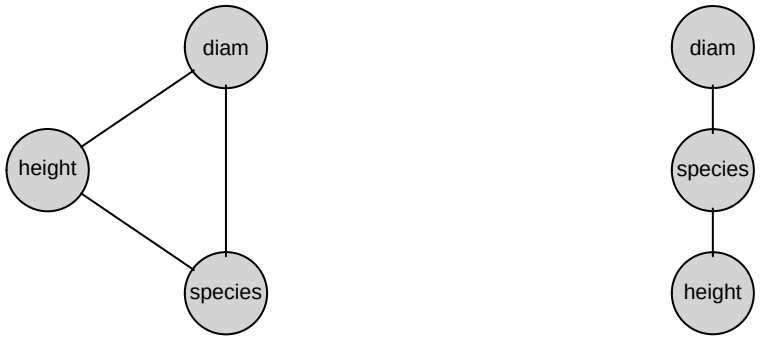
14 Log–linear models with gRim

```r
## Saturated model
liz1 <- dmod(~ height:species:diam, data=lizard)
## Shorter form
liz1 <- dmod(~.^., data=lizard)
## backward search among decomposable log-linear models;
## focus on deleting edges
liz2 <- stepwise( liz1, details=1 )
```

```
## STEPWISE:
##  criterion: aic ( k = 2 )
##  direction: backward
##  type     : decomposable
##  search   : all
##  steps    : 1000
```

```
## . BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
## . Initial model: is graphical=TRUE is decomposable=TRUE
##   change.AIC   -1.9744 Edge deleted: diam height
```

```
par(mfrow = c(1,2)); plot(liz1); plot( liz2 )
```




```
liz2
```

```
## Model: A dModel with 3 variables
##  graphical :  TRUE  decomposable :  TRUE
##  -2logL    :        1604.43 mdim :    5 aic :      1614.43
##  ideviance :          23.01 idf  :    2 bic :      1634.49
##  deviance  :           2.03 df   :    2
```

```
formula( liz2 )
```

```
## ~diam * species + height * species
```

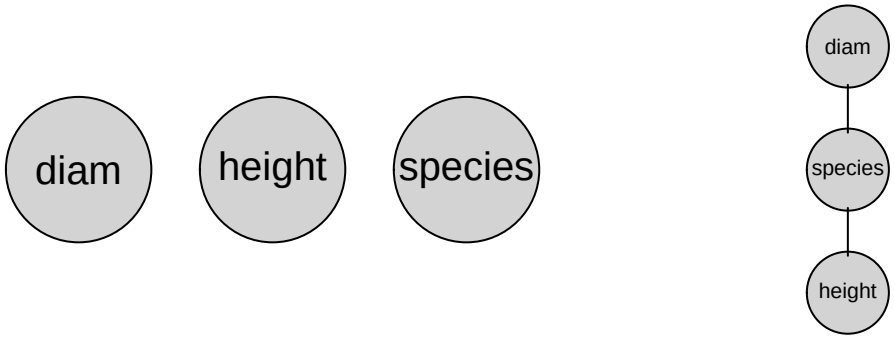
```
str( terms( liz2 ) )
```

```
## List of 2
##  $ : chr [1:2] "diam" "species"
##  $ : chr [1:2] "height" "species"
```

```
## Independence model
liz0 <- dmod(~ height + species + diam, data=lizard)
## Shorter form
liz0 <- dmod(~ .^1, data=lizard)
## forward search among decomposable log-linear models;
## focus on adding edges
liz3 <- stepwise( liz0, direction = "forward", details=1 )
```

```
## STEPWISE:
##  criterion: aic ( k = 2 )
##  direction: forward
##  type     : decomposable
##  search   : all
##  steps    : 1000
## . FORWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
## . Initial model: is graphical=TRUE is decomposable=TRUE
##   change.AIC  -10.6062 Edge added: diam species
##   change.AIC   -8.4049 Edge added: height species
```

```
par(mfrow = c(1,2)); plot(liz0); plot( liz3 )
```




```
liz3

## Model: A dModel with 3 variables
##  graphical :  TRUE  decomposable :  TRUE
##  -2logL    :        1604.43 mdim :    5 aic :      1614.43
##  ideviance :          23.01 idf  :    2 bic :      1634.49
##  deviance  :           2.03 df   :    2

formula( liz3 )

## ~diam * species + height * species

str( terms( liz3 ) )

## List of 2
##  $ : chr [1:2] "diam" "species"
##  $ : chr [1:2] "height" "species"
```

Some additional models:

```
## Saturated model:
dmod(~ .^., data=lizard)
## Independence model:
dmod(~ .^1, data=lizard)
## Decomposable graphical models {DS, HS}:
dmod(~ height:species + diam:species, data=lizard)
## Non-graphical model {DS, HS, DH}:
dmod(~ height:species + diam:species + height:diam, data=lizard)
```

14.1 Example: The reinis data

```
data(reinis, package="gRbase")
ftable(reinis, row.vars=1:3)
```

```
##                   systol    y              n
##                   protein   y       n      y       n
##                   family    y   n   y   n   y   n   y   n
## smoke mental phys
## y     y      y             44   5  23   7  35   4  24   4
##              n            129   9  50   9 109  14  51   5
##       n      y            112  21  70  14  80  11  73  13
##              n             12   1   7   2   7   5   7   4
## n     y      y             40   7  32   3  12   3  25   0
##              n            145  17  80  16  67  17  63  14
##       n      y             67   9  66  14  33   8  57  11
##              n             23   4  13   3   9   2  16   4
```

The `reinis` data: A 2^6 contingency table with risk factors of coronary heart disease. Propspective study of 1841 car–workers in Czechoslovakia, reported in 1981.

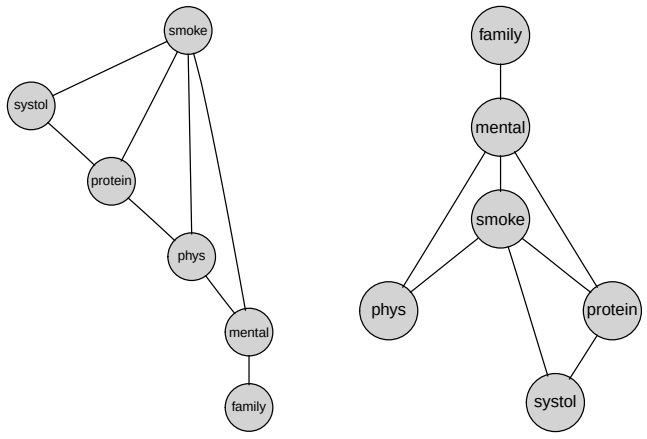
```
rei.s <- dmod(~ .^., data=reinis)
rei.s.2 <- stepwise(rei.s, details=1)
```

```
## STEPWISE:
##  criterion: aic ( k = 2 )
##  direction: backward
##  type     : decomposable
##  search   : all
##  steps    : 1000
## . BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
## . Initial model: is graphical=TRUE is decomposable=TRUE
##   change.AIC  -19.7744 Edge deleted: mental systol
```

```
##   change.AIC   -8.8511 Edge deleted: phys systol
##   change.AIC   -4.6363 Edge deleted: mental protein
##   change.AIC   -1.6324 Edge deleted: systol family
##   change.AIC   -3.4233 Edge deleted: family protein
##   change.AIC   -0.9819 Edge deleted: phys family
##   change.AIC   -1.3419 Edge deleted: smoke family
```

```
rei.1 <- dmod(~ .^1, data=reinis)
rei.1.2 <- stepwise(rei.1, direction = "forward", details=1)


## STEPWISE:
##  criterion: aic ( k = 2 )
##  direction: forward
##  type     : decomposable
##  search   : all
##  steps    : 1000
## . FORWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
## . Initial model: is graphical=TRUE is decomposable=TRUE
##   change.AIC -683.9717 Edge added: mental phys
##   change.AIC  -25.4810 Edge added: smoke phys
##   change.AIC  -15.9293 Edge added: protein mental
##   change.AIC  -10.8092 Edge added: systol protein
##   change.AIC   -2.7316 Edge added: family mental
##   change.AIC   -1.9876 Edge added: mental smoke
##   change.AIC  -16.4004 Edge added: protein smoke
##   change.AIC  -12.5417 Edge added: systol smoke
```

```
par(mfrow=c(1,2)); plot(rei.s.2); plot(rei.1.2)
```



14.2 Example: From graphical model to BN

```
bn.s.2 <- grain(rei.s.2)
bn.s.2
```

```
## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:6] "smoke" "protein" "systol" "phys" "mental" ...
```

14.3 Example: The coronary artery disease data

CAD is the disease; the other variables are risk factors and disease manifestations/symptoms.

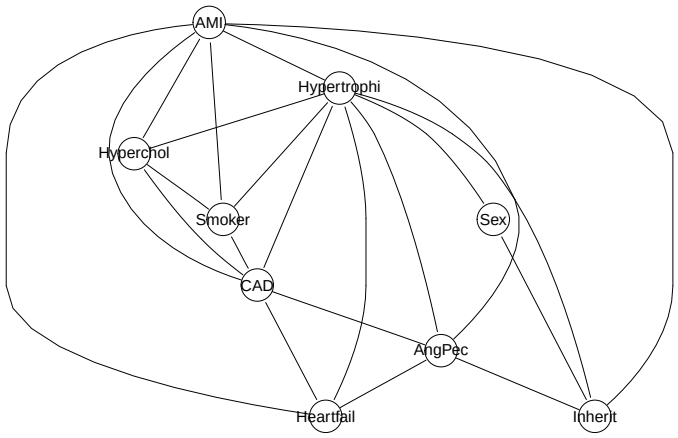
```
data(cad1, package="gRbase")
use  <- c(1,2,3,9:14)
cad1 <- cad1[,use]
head( cad1, 4 )
```

```
##      Sex   AngPec        AMI Hypertrophi Hyperchol Smoker Inherit
## 1   Male     None NotCertain          No        No     No      No
## 2   Male Atypical NotCertain          No        No     No      No
## 3 Female     None   Definite          No        No     No      No
## 4   Male     None NotCertain          No        No     No      No
##   Heartfail CAD
## 1        No  No
## 2        No  No
## 3        No  No
## 4        No  No
```

```
m.sat <- dmod( ~.^., data=cad1 ) # saturated model
m.new1 <- stepwise( m.sat, details=0, k=2 ) # use aic
m.new1
```

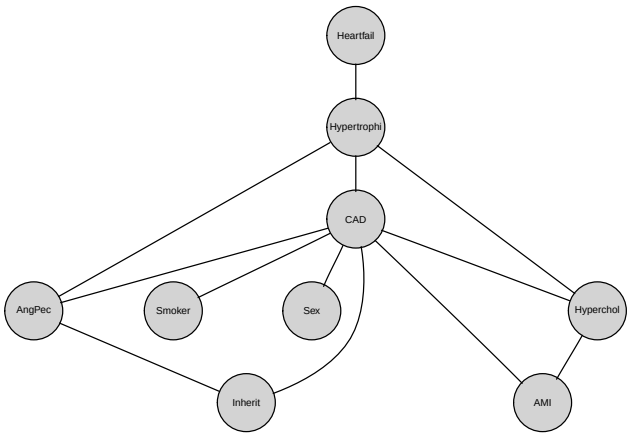
```
## Model: A dModel with 9 variables
##  graphical :  TRUE  decomposable :  TRUE
##  -2logL    :        2275.24 mdim :   87 aic :      2449.24
##  ideviance :         425.03 idf  :   77 bic :      2750.59
##  deviance  :         227.02 df   :  680
## Notice: Table is sparse
##   Asymptotic chi2 distribution may be questionable.
##   Degrees of freedom can not be trusted.
##   Model dimension adjusted for sparsity : 60
```

```
## plot( m.new1 ) ## Works but unreadable
at <- list(node=list(fontsize=40))
plot(ug(terms(m.new1)), attrs=at)
```



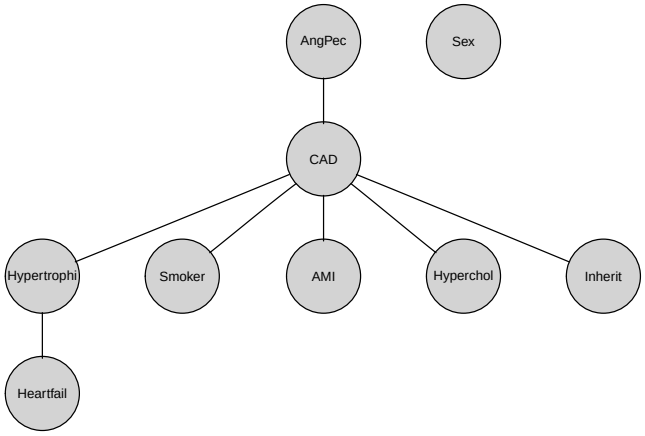

```
m.ind <- dmod( ~ .^1, data=cad1 ) # independence model
m.new2 <- stepwise( m.ind, direction="forward", details=0, k=2 )
m.new2


## Model: A dModel with 9 variables
##  graphical :  TRUE  decomposable :  TRUE
##  -2logL    :        2379.06 mdim :   31 aic :      2441.06
##  ideviance :         321.21 idf  :   21 bic :      2548.44
##  deviance  :         330.84 df   :  736
```

```
plot( m.new2 )
```



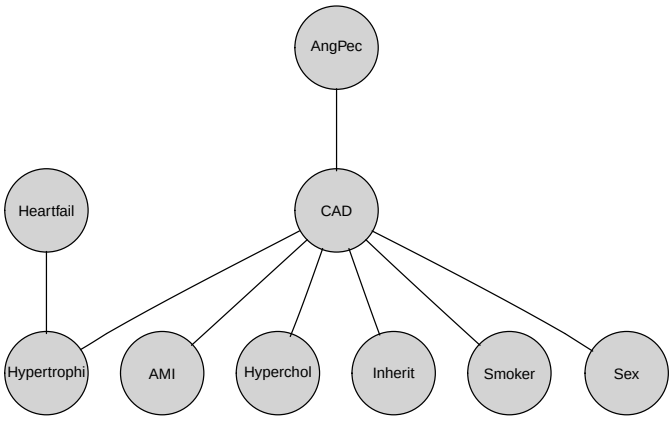
```
m.new3 <- stepwise( m.sat,  k=log(nrow(cad1)) ) # use bic
m.new3


## Model: A dModel with 9 variables
##  graphical :  TRUE  decomposable :  TRUE
##  -2logL    :        2420.32 mdim :   18 aic :      2456.32
##  ideviance :         279.94 idf  :    8 bic :      2518.67
##  deviance  :         372.11 df   :  749
```

```
plot( m.new3 )
```




```
m.new4 <- stepwise( m.ind, direction="forward", k=log(nrow(cad1)) ) # use bic
```

```
plot( m.new4 )
```



14.4 From graph and data to network

Create Bayesian networks from model2 (i.e. from (graph,data)):

`smooth`: Add a small number to each cell to avoid configurations with zero probability.

```
bn1 <- grain( m.new1, smooth=0.01 )
bn2 <- grain( m.new2, smooth=0.01 )
bn3 <- grain( m.new3, smooth=0.01 )
bn4 <- grain( m.new4, smooth=0.01 )
qgrain( bn1, "CAD")$CAD
```

```
## CAD
##    No   Yes
## 0.547 0.453
```

```
qgrain( bn1, "CAD",
          evidence=list(AngPec="Typical", Hypertrophi="Yes"))$CAD
```

```
## CAD
##  No Yes
## 0.6 0.4
```

15 Prediction

Dataset with missing values

```
data(cad2, package="gRbase")
use  <- c(1,2,3,9:14)
cad2 <- cad2[,use]
head( cad2, 4 )
```

```
##      Sex   AngPec        AMI Hypertrophi Hyperchol Smoker Inherit
## 1   Male     None NotCertain          No         No   <NA>      No
## 2 Female     None NotCertain          No         No   <NA>      No
## 3 Female     None NotCertain          No        Yes   <NA>      No
## 4   Male Atypical   Definite          No        Yes   <NA>      No
##   Heartfail CAD
## 1        No  No
## 2        No  No
## 3        No  No
## 4        No  No
```

```
p1 <- predict(bn1, newdata=cad2, response="CAD")
head( p1$pred$CAD )

## [1] "No"  "No"  "No"  "No"  "No"  "Yes"
```

```
z <- data.frame(CAD.obs=cad2$CAD, CAD.pred=p1$pred$CAD)
head( z ) # class assigned by highest probability

##   CAD.obs CAD.pred
## 1      No       No
## 2      No       No
## 3      No       No
## 4      No       No
## 5      No       No
## 6      No      Yes

xtabs(~., data=z)

##        CAD.pred
## CAD.obs No Yes
##     No  32   9
##     Yes 10  16
```

Let us do so for all models

```
p <-
    lapply(list(bn1, bn2, bn3, bn4),
       function(bn) predict(bn, newdata=cad2, response="CAD"))
l <- lapply(p, function(x) x$pred$CAD)
cls <- as.data.frame(l)
names(cls) <- c("CAD.pred1","CAD.pred2","CAD.pred3","CAD.pred4")
cls$CAD.obs <- cad2$CAD
head(cls)

##   CAD.pred1 CAD.pred2 CAD.pred3 CAD.pred4 CAD.obs
## 1        No        No        No        No      No
## 2        No        No        No        No      No
## 3        No        No        No        No      No
## 4        No        No       Yes       Yes      No
## 5        No        No        No        No      No
## 6       Yes        No        No        No      No
```

```
xtabs( ~ CAD.obs+CAD.pred1, data=cls)
```

```
##         CAD.pred1
## CAD.obs No Yes
##     No  32   9
##     Yes 10  16
```

```
xtabs( ~ CAD.obs+CAD.pred2, data=cls)
```

```
##         CAD.pred2
## CAD.obs No Yes
##     No  35   6
##     Yes 10  16
```

```
xtabs( ~ CAD.obs+CAD.pred3, data=cls)
```

```
##         CAD.pred3
## CAD.obs No Yes
##     No  32   9
##     Yes 10  16
```

```
xtabs( ~ CAD.obs+CAD.pred4, data=cls)
```

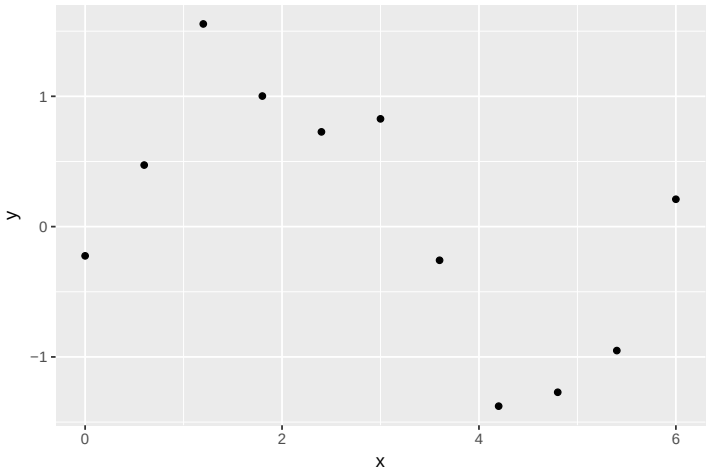
```
##         CAD.pred4
## CAD.obs No Yes
##     No  33   8
##     Yes 10  16
```

16 Further topics

16.1 Discretization and its curse

Consider this example:

```
qplot(x, y)
```



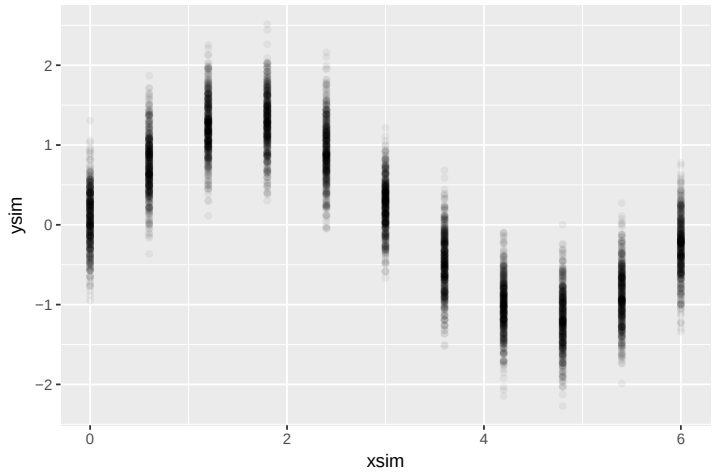
Suppose $y|x \sim N(\sin(x), \sigma^2)$. Approximate $x|y$ by discrete version:

```
yb <- seq(-2, 2, 0.5)
xb <- seq(0, 2*pi, 0.6)

yd <- cut(y, breaks=yb)
xd <- cut(x, breaks=xb)

xb <- xb[-1] - 0.3
levels(xd) <- xb

df <- data.frame(y=yd,x=xd)
df <- df[complete.cases(df),]
tab <- xtabs(~y+x, data=df)
tab <- gRbase::ar_normalize(tab, type="first")
round(100 * tab)
```

```
##             x
## y            0.3 0.9 1.5 2.1 2.7 3.3 3.9 4.5 5.1 5.7
##   (-2,-1.5]   0   0   0   0   0   0   0   0   0   0
##   (-1.5,-1]   0   0   0   0   0   0 100 100   0   0
##   (-1,-0.5]   0   0   0   0   0   0   0   0 100   0
##   (-0.5,0]    0   0   0   0   0 100   0   0   0   0
##   (0,0.5]   100   0   0   0   0   0   0   0   0 100
##   (0.5,1]     0   0   0 100 100   0   0   0   0   0
##   (1,1.5]     0   0 100   0   0   0   0   0   0   0
##   (1.5,2]     0 100   0   0   0   0   0   0   0   0
```

Probably (??) better to simulate base approximation on simulation under the model:

```
m <- lm(y ~ sin(x))
Nsim <- 500
ysim <- as.numeric(as.matrix(simulate(m, n=Nsim)))
xsim <- rep(x, Nsim)
ggplot(data.frame(xsim,ysim),aes(x=xsim, y=ysim)) + geom_point(alpha = 0.05)
```




```
yd <- cut(ysim, breaks=yb)
xd <- cut(xsim, breaks=xb)

xb <- xb[-1] - 0.3
levels(xd) <- xb

df <- data.frame(y=yd,x=xd)
df <- df[complete.cases(df),]
tab <- xtabs(~y+x, data=df)
tab <- gRbase::ar_normalize(tab, type="first")
round(tab*100)
```

```
##             x
## y            0.6 1.2 1.8 2.4  3 3.6 4.2 4.8 5.4
##   (-2,-1.5]   0   0   0   0  0   0   7  18   5
##   (-1.5,-1]   0   0   0   0  0   7  45  49  33
##   (-1,-0.5]   0   0   0   0  1  41  42  30  50
##   (-0.5,0]    2   0   0   1 20  43   6   3  12
##   (0,0.5]    22   2   2  11 59   8   0   0   1
##   (0.5,1]    53  23  18  46 19   0   0   0   0
##   (1,1.5]    21  53  54  35  1   0   0   0   0
##   (1.5,2]     2  22  26   7  0   0   0   0   0
```

In either case, we get an approximation to $y|x$ as an array, but the array has many entries of which many are zero.

16.2 Hard and virtual (likelihood) evidence

Consider the following excerpt of the chest clinic network.

```
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 ) ) )


## CPTspec with probabilities:
##  P( asia )
##  P( tub | asia )


( chest1 <- grain(plist1) )


## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:2] "asia" "tub"


qgrain( chest1 )


## $asia
## asia
##  yes   no
## 0.01 0.99
##
## $tub
## tub
##    yes     no
## 0.0104 0.9896
```

16.3 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 `setEvi()` can both be used for this purpose. The following forms are equivalent.

```
setEvidence( chest1, evidence=list(asia="yes"))
setEvidence( chest1, nodes="asia", states="yes")
```

```
qgrain( setEvi( chest1, evidence=list(asia="yes")) )


## $tub
## tub
##  yes   no
## 0.05 0.95
```

16.4 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.

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

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


##           asia
## guess.asia yes  no
##        yes 0.8 0.1
##        no  0.2 0.9
## attr(,"class")
## [1] "parray" "array"
```

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 Asia we would (erroneously) guess that they have recently been to Asia in 10% of the cases.

```
( plist2 <- compileCPT( list( a, t.a, g.a ) ) )


## CPTspec with probabilities:
##  P( asia )
##  P( tub | asia )
##  P( guess.asia | asia )
```

```
( chest2 <- grain(plist2) )

## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr [1:3] "asia" "tub" "guess.asia"

qgrain( 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:

```
qgrain( setEvi( chest2, evidence=list(guess.asia="yes")) )

## $asia
## asia
##    yes     no
## 0.0748 0.9252
##
## $tub
## tub
##   yes    no
## 0.013 0.987
```

16.5 Specifying virtual evidence

However, it is NOT necessary to do so in practice, because we may equivalently enter the virtual evidence in the original network.

```
qgrain( setEvi( chest1, evidence=list(asia=c(.8, .1))) )


## $tub
## tub
##   yes    no
## 0.013 0.987
```

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

```
qgrain( setEvi( chest1, evidence=list(asia=c(1, 0))) )


## $tub
## tub
##  yes   no
## 0.05 0.95
```

16.6 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.

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

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

```
p.x <- ar_new(~X, levels=list(X=1:2), values=c(.25, .75))
bn.simple <- grain(compileCPT(list(p.x)))
bn.simple


## Independence network: Compiled: FALSE Propagated: FALSE
##   Nodes: chr "X"
```

```
qgrain(bn.simple)
```

```
## $X
## X
##    1    2
## 0.25 0.75
```

Suppose $y_1|x = k \sim N(\mu_k, \sigma_k^2)$ and $y_2|x = k \sim Poi(\lambda_k)$ where $k = 1, 2$.

```
y1 <- 1.3
y2 <- 5
mu <- c(1, 4)
sigma <- 1
lambda <- c(2, 8)
L1 <- dnorm(y1, mean=mu, sd=sigma)
L2 <- dpois(y2, lambda=lambda)
L <- L1 * L2
L1
```

```
## [1] 0.3814 0.0104
```

```
L2
```

```
## [1] 0.0361 0.0916
```

```
L
```

```
## [1] 0.013764 0.000955
```

Now we can enter this likelihood evidence

```
bn.simple.ev <- setEvi(bn.simple, evidence=list(X=L))
qgrain( bn.simple.ev, exclude=F )
```

```
## $X
## X
##     1     2
## 0.828 0.172
```

16.7 Disclaimer: Bayesian networks, Bayes' formula and Bayesian statistics

Wikipedia (`https://en.wikipedia.org/wiki/Bayesian_network`) says

> *The term "Bayesian networks" was coined by Judea Pearl in 1985 to emphasize three aspects:*
>
> 1. *The often subjective nature of the input information.*
> 2. *The reliance on Bayes' conditioning as the basis for updating information.*
> 3. *The distinction between causal and evidential modes of reasoning, which underscores Thomas Bayes' posthumously published paper of 1763.*

Bayesian networks are based on Bayes' formula 1) when we build a network and 2) when we make inference in the network.

A common misunderstanding is that Bayesian networks are related to Bayesian statistics, and that is not really the case: We may obtain the components of a Bayesian network (or the whole network itself) from a Bayesian statistical analysis, or from a purely frequentist analysis.

17 Winding up

Brief summary:

- We have gone through aspects of the **gRain** package and seen some of the mechanics of probability propagation.
- Propagation is based on factorization of a pmf according to a decomposable graph.
- We have gone through aspects of the **gRim** package and seen how to search for decomposable graphical models.
- We have seen how to create a Bayesian network from the dependency graph of a decomposable graphical model.

- The model search facilities in **gRim** do not scale to large problems; instead it is more useful to consider other approaches for structural learning, see e.g. the **bnlearn** package

Index