Graphical Models and Bayesian Networks

– Guanajuato, México, 2015

Søren Højsgaard

Department of Mathematical Sciences

Aalborg University, Denmark

February 10, 2015

Printed: February 10, 2015 File: bayesnet-slides.tex

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 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"))
```

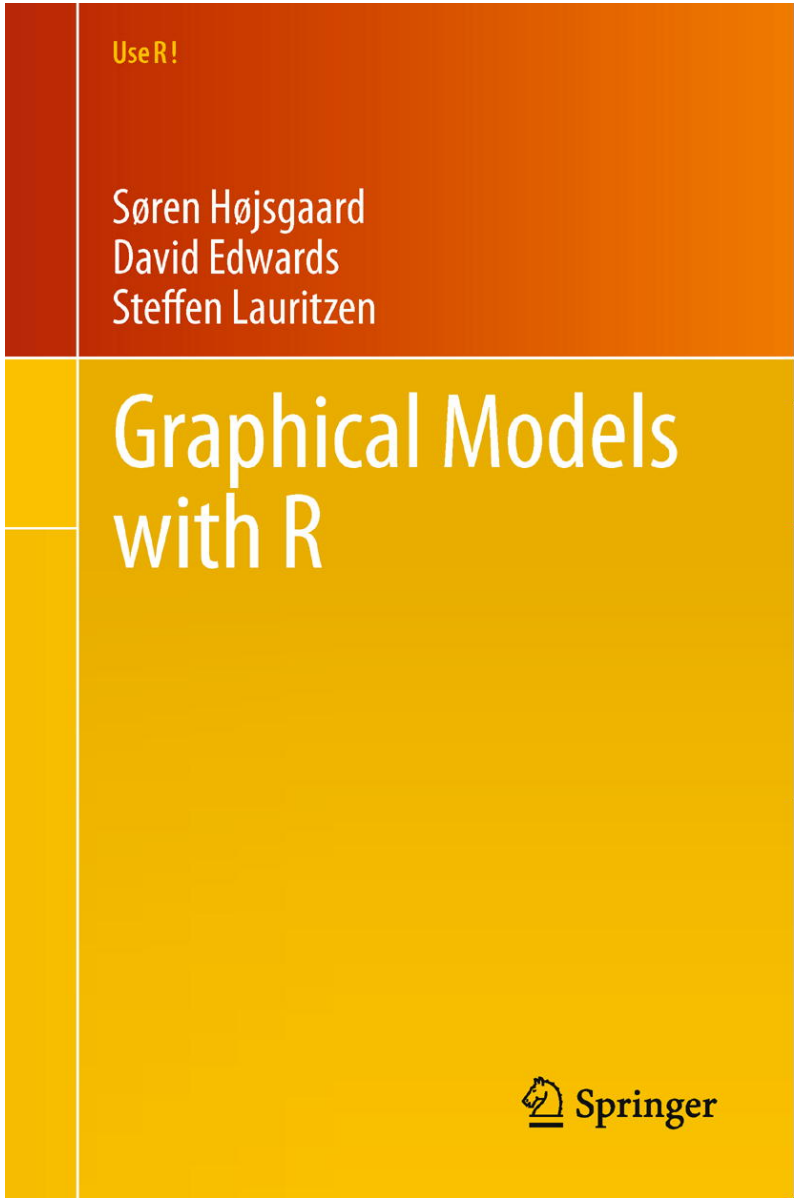
Then install the packages from CRAN.

Tutorial based on these development versions:

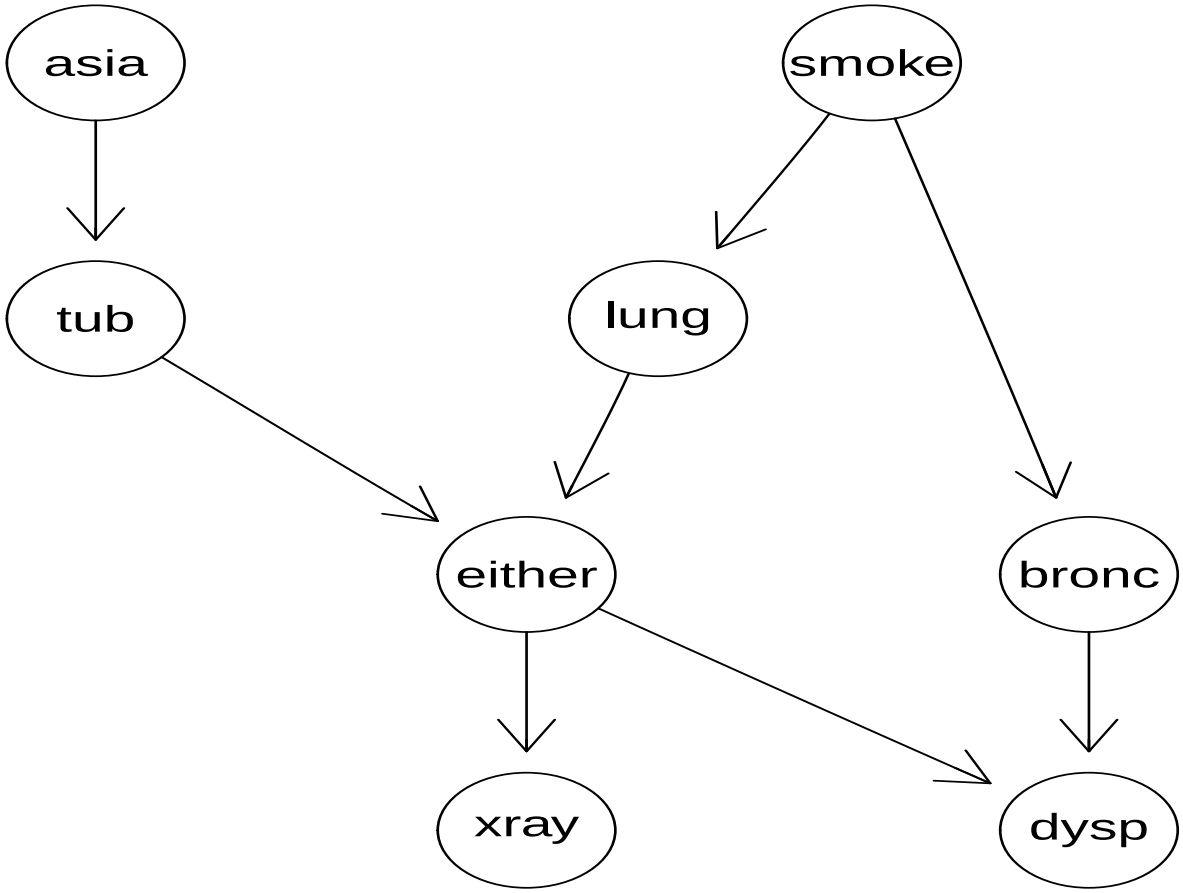
```
> packageVersion("gRbase")
[1] '1.7.2'
> packageVersion("gRain")
[1] '1.2.4'
> packageVersion("gRim")
[1] '0.1.18'
```

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

1.2 Book: Graphical Models with R



2 The chest clinic narrative

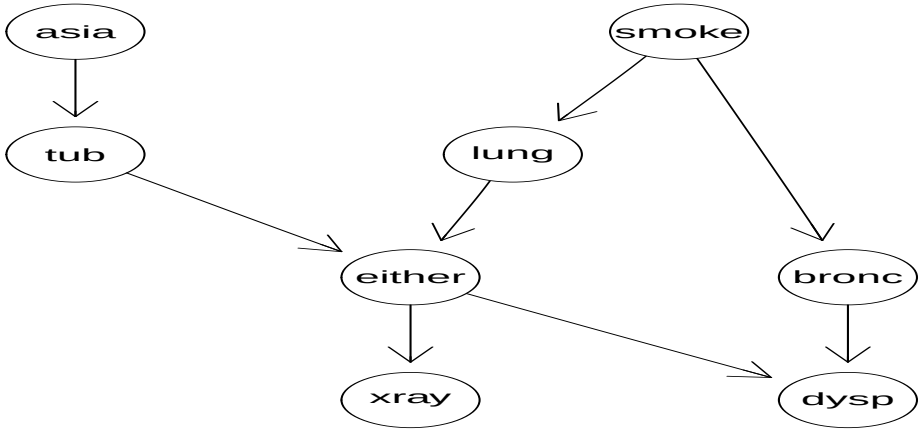


Lauritzen and Spiegehalter (1988) present the following narrative:

- "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)

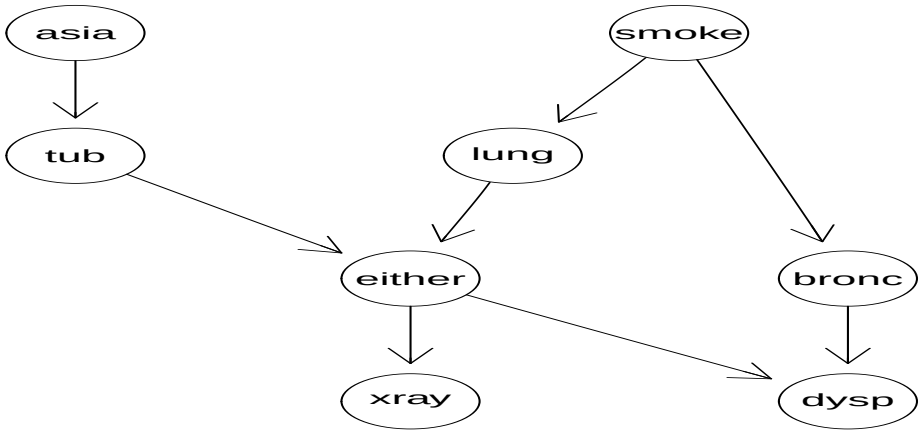
2.1 DAG–based models



- Each node v represents a random variable Z_v (here binary with levels "yes","no".
- The nodes

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

correspond to 8–dim random vector $Z_V = (Z_a, \ldots, Z_d)$.



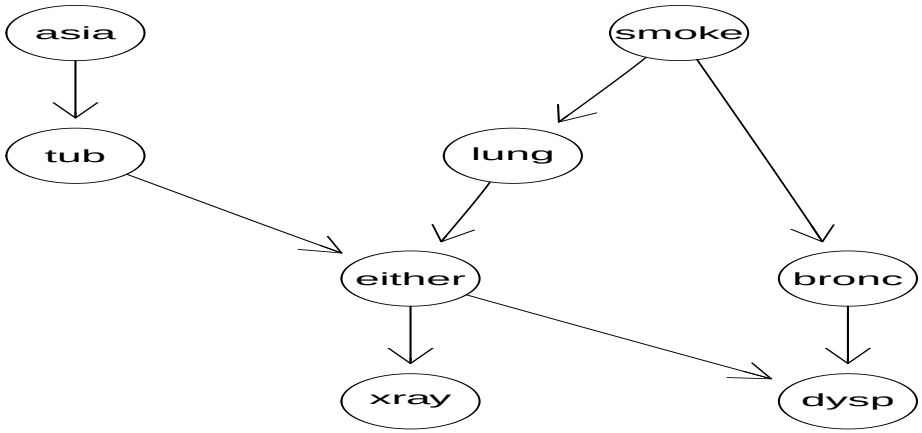
- We want to specify probability density

$$p_{Z_V}(z_V) \text{ or shorter } p(V)$$

- For each combination of a node v and its parents $pa(v)$ there is a conditional distribution $p(z_v|z_{pa(v)})$, for example

$$p_{Z_e|Z_t,Z_l}(z_{either}|z_{tub}, z_{lung}) \text{ or shorter } p(e|t, l)$$

- Specified as a conditional probability table (a CPT), for example for $p(e|t, l)$ the CPT is a $2 \times 2 \times 2$–table



- Recall: Allow for informal notation: Write $p(V)$ instead of $p_V(z_V)$; write $p(v|pa(v))$ instead of $p(z_v|z_{pa(v)})$.
- The DAG corresponds to a factorization of the joint probability function as

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

3 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: parray() from gRbase
> t.a <- parray(c("tub","asia"), levels=list(yn,yn), values=x)
> t.a <- parray(~tub:asia, levels=list(yn,yn), values=x) # alternativ
> 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.

4 An introduction to the **gRain** package

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

```
> library(gRain)
> 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" ...
```

5 Querying the network

```
> # Query network to find marginal probabilities of diseases
> querygrain(bn, nodes=c("tub","lung","bronc"))
$tub
tub
   yes     no
0.0104 0.9896

$lung
lung
  yes    no
0.055 0.945

$bronc
bronc
 yes   no
0.45 0.55
```

6 Setting evidence

```
> # Set evidence and query network again
> bn.ev <- setEvidence(bn, nslist=list(asia="yes",dysp="yes"))
> querygrain(bn.ev, nodes=c("tub","lung","bronc"))
$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
[1] "asia" "dysp"

$states
$states$asia
[1] "yes"

$states$dysp
[1] "yes"


attr(,"pFinding")
[1] 0.0045
> # Probability of observing the evidence (the normalizing constant)
> pEvidence(bn.ev)
[1] 0.0045
```

```
> # Set additional evidence and query again
> bn.ev <- setEvidence(bn.ev, nslist=list(xray="yes"))
> querygrain(bn.ev, nodes=c("tub","lung","bronc"))
$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 <- querygrain(bn.ev, nodes=c("tub","lung","bronc"),
                  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
> # Remove evidence
> bn.ev<-retractEvidence(bn.ev, nodes="asia")
> bn.ev
Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
  Findings: chr [1:2] "dysp" "xray"
```

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:

```
> querygrain(bn, nslist=list(asia="yes",dysp="yes"),
                      nodes=c("tub","lung","bronc"))
$tub
tub
   yes     no
0.0878 0.9122

$lung
lung
   yes     no
0.0995 0.9005

$bronc
bronc
  yes    no
0.811 0.189
```

7 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 $p(v|pa(v))$:

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

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.

```
> ## collection of CPTs: p(v|pa(v))
> 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 )
> ## form joint p(V)= prod p(v|pa(v))
> joint <- tableListProd( cpt.list )
> dim( joint )
  dysp  bronc either   xray   asia   lung  smoke    tub
     2      2      2      2      2      2      2      2
> head( as.data.frame.table( joint ), 4 )
  dysp bronc either xray asia lung smoke tub     Freq
1  yes   yes    yes  yes  yes  yes   yes yes 1.32e-05
2   no   yes    yes  yes  yes  yes   yes yes 1.47e-06
3  yes    no    yes  yes  yes  yes   yes yes 6.86e-06
4   no    no    yes  yes  yes  yes   yes yes 2.94e-06
```

```
> ## form marginal p(a,b,d) by marginalization
> marg <- tableMargin(joint, ~asia+bronc+dysp)
> dim( marg )
 asia bronc  dysp
    2     2     2
> ftable( marg )
            dysp      yes        no
asia bronc
yes  yes         0.003652 0.000848
     no          0.000849 0.004651
no   yes         0.359933 0.085567
     no          0.071536 0.472964
```

```
> ## Set entries not consistent with asia=yes and dysp=yes to zero
> marg <- setSliceValue(marg, slice=list(asia="yes",dysp="yes"),
                        complement=T, value=0)
> ftable(marg)
           dysp      yes       no
asia bronc
yes  yes         0.003652 0.000000
     no          0.000849 0.000000
no   yes         0.000000 0.000000
     no          0.000000 0.000000
> result <- tableMargin(marg, ~bronc);
> result <- result / sum( result ); result
bronc
  yes    no
0.811 0.189
>
```

7.1 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?

The trick is to NOT to calculate the joint distribution

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

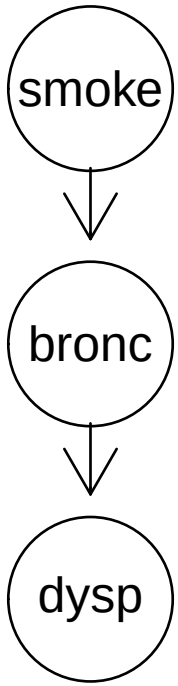
explicitly because that leads to working with high dimensional tables.

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

The challenge is to organize these local computations.

8 Message passing – a small example

```
> d<-dag( ~smoke + bronc|smoke + dysp|bronc ); plot(d)
```



Joint distribution

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

```
> yn <- c("yes","no")
> s   <- parray("smoke", list(yn), c(.5, .5))
> b.s <- parray(c("bronc","smoke"), list(yn,yn), c(6,4, 3,7))
> d.b <- parray(c("dysp","bronc"), list(yn, yn), c(9,1, 2,8))
> cpt.list <- list(s, b.s, d.b); cpt.list
[[1]]
smoke
yes  no
0.5 0.5

[[2]]
     smoke
bronc yes no
  yes   6  3
  no    4  7

[[3]]
     bronc
dysp  yes no
  yes   9  2
  no    1  8
```

Recall that the joint distribution is

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

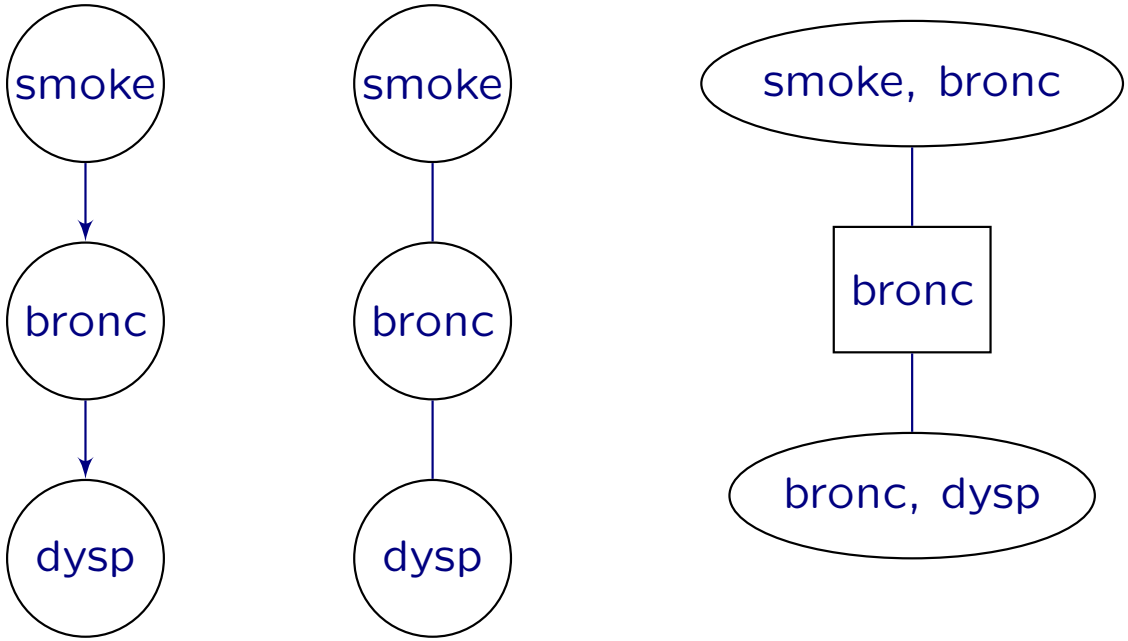
i.e.

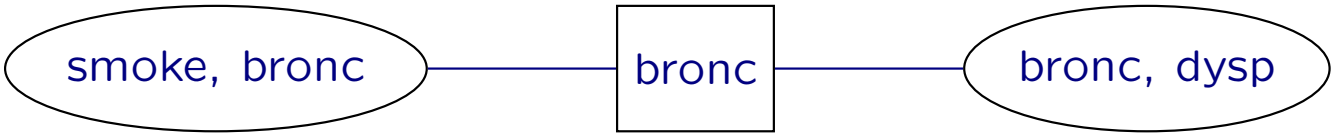
```
> joint <- tableListProd( cpt.list ) ; ftable(joint)
            smoke  yes   no
dysp bronc
yes  yes          27.0 13.5
     no            4.0  7.0
no   yes           3.0  1.5
     no           16.0 28.0
```

but we really do not want to calculate this in general; here we just do it as "proof of concept".

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

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

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

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

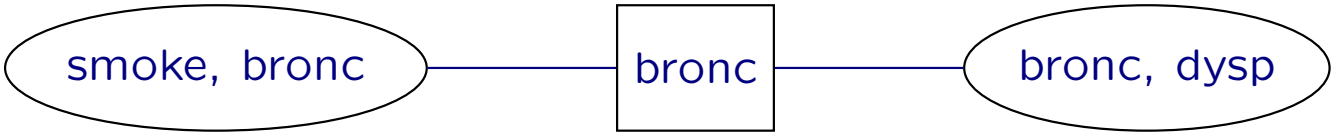
is called a clique potential representation.

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

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

```
> q1.sb <- tableMult(s, b.s); q1.sb
      smoke
bronc yes  no
  yes   3 1.5
  no    2 3.5
> q2.bd <- d.b; q2.bd
      bronc
dysp  yes no
  yes   9  2
  no    1  8
```

8.1 Collect Evidence



We pick any node, say $(bronc, dysp) = (b, d)$ as root in the junction tree, and work inwards towards the root as follows.

First, define $q_1(b) \leftarrow \sum_s q_1(s, b)$.

We have

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

Therefore, we update potentials as

$$q_1(s, b) \leftarrow q_1(s, b)/q_1(b), \quad q_2(b, d) \leftarrow q_2(b, d)q_1(b)$$

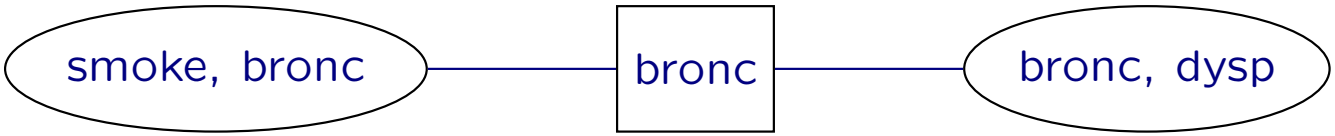
and the new potentials are also defined on the cliques of the junction tree. We still have

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

Updating of potentials is done as follows:

```
> q1.b  <- tableMargin(q1.sb, "bronc"); q1.b
bronc
yes  no
4.5 5.5
> q2.bd <- tableMult(q2.bd, q1.b); q2.bd
     dysp
bronc  yes   no
  yes 40.5  4.5
  no  11.0 44.0
> q1.sb <- tableDiv(q1.sb, q1.b); q1.sb
     smoke
bronc   yes    no
  yes 0.667 0.333
  no  0.364 0.636
```

8.2 Distribute Evidence



Next work outwards from the root (b, d).

Set $q_2(b) \leftarrow \sum_d q_2(b, d)$. We have

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

We therefore update as $q_1(s, b) \leftarrow q_1(s, b)q_2(b)$ and have

$$p(s, b, d) = q_1(s, b)q_2(b, d) = \frac{q_1(s, b)q_2(b, d)}{q_2(b)}$$

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

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

```
> q2.b <- tableMargin(q2.bd, "bronc"); q2.b
bronc
yes  no
 45  55
> q1.sb <- tableMult(q1.sb, q2.b); q1.sb
      smoke
bronc yes no
  yes  30 15
  no   20 35
```

Recall that the joint distribution is

```
> joint
, , smoke = yes

     bronc
dysp  yes no
  yes  27  4
  no    3 16

, , smoke = no

     bronc
dysp   yes no
  yes 13.5  7
  no   1.5 28
```

Claim: After these steps $q_1(s,b) = p(s,b)$ and $q_2(b,d) = p(b,d)$.

Proof:

```
> q1.sb
      smoke
bronc yes no
  yes  30 15
  no   20 35
> tableMargin(joint, c("smoke","bronc"))
      bronc
smoke yes no
  yes  30 20
  no   15 35
> q2.bd
      dysp
bronc  yes   no
  yes 40.5  4.5
  no  11.0 44.0
> tableMargin(joint, c("bronc","dysp"))
      dysp
bronc  yes   no
  yes 40.5  4.5
  no  11.0 44.0
```

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

```
> tb <- tableMargin(q1.sb, "bronc")
> tb/sum( tb )
bronc
 yes   no
0.45 0.55
> tb <- tableMargin(q2.bd, "bronc") # or
> tb/sum( tb )
bronc
 yes   no
0.45 0.55
```

And we NEVER calculated the full joint distribution!

8.3 Setting evidence

Next consider the case where we have the evidence that dysp=yes.

```
> q1.sb <- tableMult(s, b.s)
> q2.bd <- d.b
> q2.bd <- setSliceValue(q2.bd, list(dysp="yes"), comp=T); q2.bd
      bronc
dysp   yes no
  yes    9  2
  no     0  0
> # Repeat all the same steps as before
> q1.b  <- tableMargin(q1.sb, "bronc")
> q2.bd <- tableMult(q2.bd, q1.b)
> q1.sb <- tableDiv(q1.sb, q1.b)
> q2.b  <- tableMargin(q2.bd, "bronc")
> q1.sb <- tableMult(q1.sb, q2.b)
```

Claim: After these steps $q_1(s, b) = p(s, b|d^+)$ and $q_2(b, d) = p(b, d|d^+)$.

```
> joint <- setSliceValue(joint, list(dysp="yes"), comp=T);
> ftable( joint, row.vars=1)
     bronc  yes         no
     smoke  yes   no  yes   no
dysp
yes        27.0 13.5  4.0  7.0
no          0.0  0.0  0.0  0.0
```

Proof:

```
> q1.sb
     smoke
bronc yes   no
  yes  27 13.5
  no    4  7.0
> tableMargin(joint, c("smoke","bronc"))
     bronc
smoke  yes no
  yes 27.0  4
  no  13.5  7
```

And we NEVER calculated the full joint distribution!

9 Message passing – the bigger picture

The DAG is only used in connection with specifying the network; afterwards all computations are based on properties of a derived undirected graph.

Recall goal: Avoid working with high dimensional tables.

Think of the CPTs as potentials/interactions (q-functions):

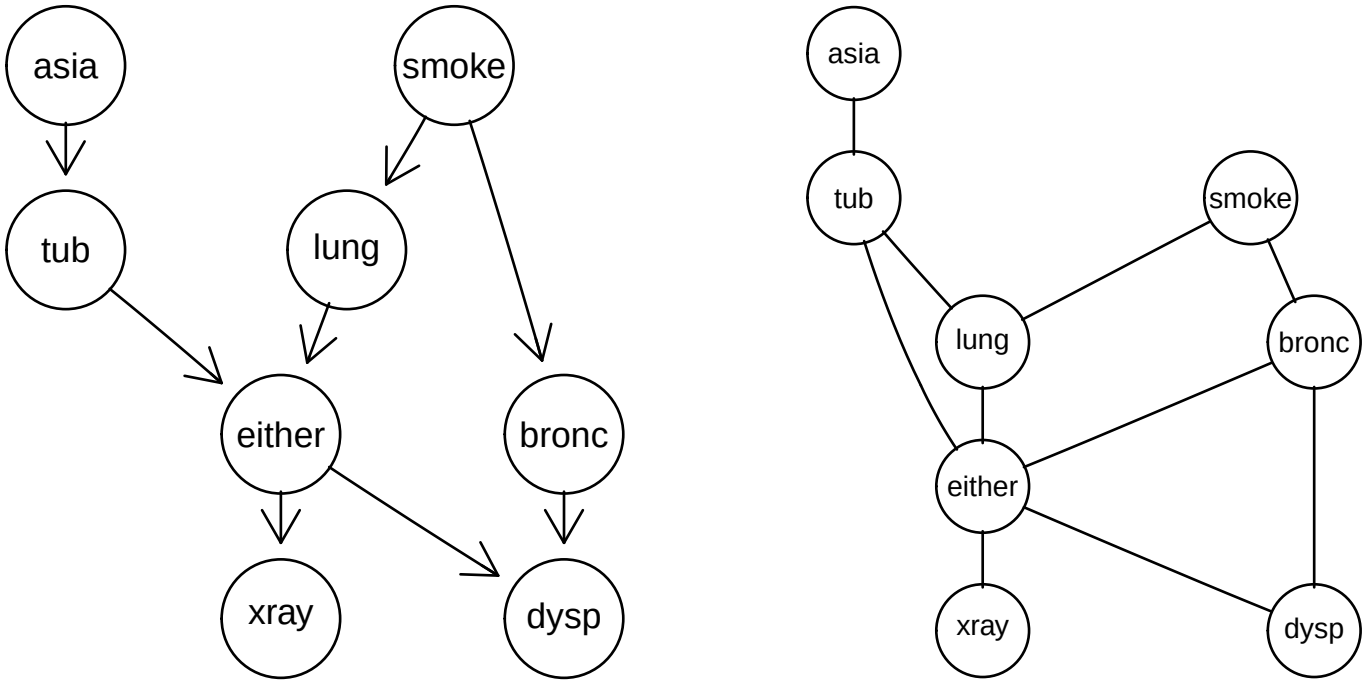
$$\begin{aligned} p(V) &= p(a)p(t|a)p(s)p(l|s)p(b|s)p(e|t,l)p(d|e,b)p(x|e) \\ &= q(a)q(t,a)q(s)q(l,s)q(b,s)q(e,t,l)q(d,e,b)q(x,e). \end{aligned}$$

Notice: q–functions that are "contained" in other q–functions can be absorbed into these; we set $q(t,a) \leftarrow q(t,a)q(a)$ and $q(l,s) \leftarrow q(l,s)q(s)$:

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

Moral graph: marry parents and drop directions:

```
> par(mfrow=c(1,2)); plot(bn$dag); plot(moralize(bn$dag))
```

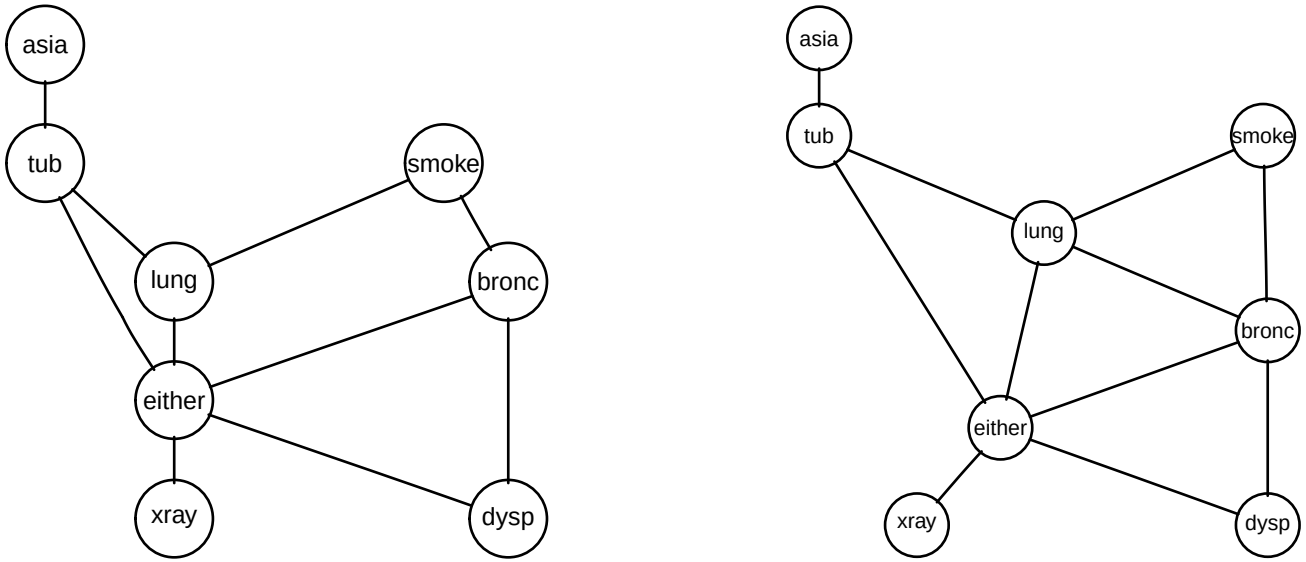


$$p(V) = q(t,a)q(l,s)q(b,s)q(e,t,l)q(d,e,b)q(x,e).$$

Notice: $p(V)$ has interactions only among neighbours of the undirected moral graph.

Efficient computations hinges on the undirected graph being chordal. We make moral graph chordal by adding fill-ins.

```
> par(mfrow=c(1,2)); plot(moralize(bn$dag));
> plot(triangulate(moralize(bn$dag)))
```



We have $p(V)$ factoring according to this chordal graph as

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

where $q(l,s,b) = q(l,s)q(b,s)$ and $q(l,b,e) \equiv 1$.

We have $p(V) = \prod_{C:cliques} q(C)$.

We want to manipulate the q–functions such that $p(C) = q(C)$ without creating high–dimensional tables.

The manipulations are of the form (where $S \subset C$)

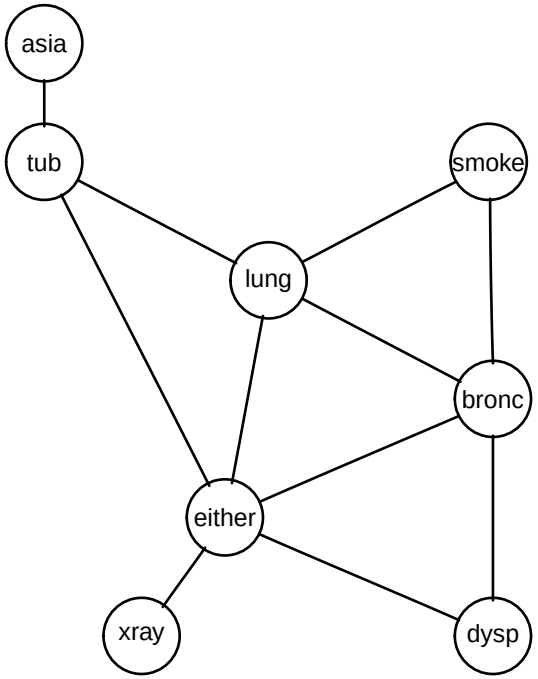
$$q(S) = \sum_{C \setminus S} q(C), \quad q(C) \leftarrow q(C)\tilde{q}(S), \quad q(C) \leftarrow q(C)/\tilde{q}(S),$$

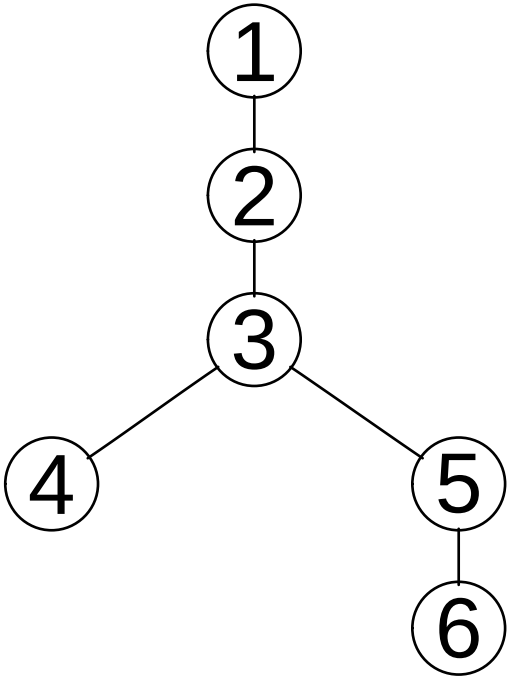
Cliques of chordal graph can be ordered such that

$$B_k = (C_1 \cup, \ldots, \cup C_{k-1}), \quad S_k = B_k \cap C_k \subset C_j \text{ for some } j < k$$

so after computing $q(S_k) = \sum_{C_k \setminus S_k} q(C_k)$ we can absorb $q(S_k)$ into a C_j by $q(C_j)q(S_k)$ which will still be a function of C_j only.

```
> par(mfrow=c(1,2)); plot(bn$ug); plot(jTree( bn$ug ))
> str( jTree( bn$ug )$cliques )
List of 6
 $ : chr [1:2] "asia" "tub"
 $ : chr [1:3] "either" "lung" "tub"
 $ : chr [1:3] "either" "lung" "bronc"
 $ : chr [1:3] "smoke" "lung" "bronc"
 $ : chr [1:3] "either" "dysp" "bronc"
 $ : chr [1:2] "either" "xray"
```

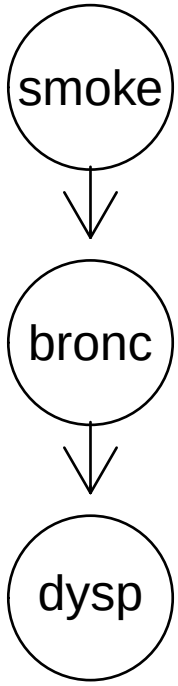




10 Conditional independence

Consider again the toy example:

```
> plot(dag(~smoke+bronc|smoke+dysp|bronc))
```



with

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

The factorization implies a conditional independence restriction:

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

or equivalently

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

Consider $p(s|b,d)$:

$$p(s|b,d) = \frac{p(s)p(b|s)p(d|b)}{\sum_s p(s)p(b|s)p(d|b)} = \frac{p(s)p(b|s)}{\sum_s p(s)p(b|s)}$$

Hence $p(s|b,d)$ is a function of (s,b) but not of d.

We say that "s is independent of d given b" or that "s and d are conditionally independent given b" and write $s \perp\!\!\!\perp d|b$.

If we know b then getting to know also b provides no additional information about s.

Conditional independence can often be deduced easier as follows: Suppose that for non–negative functions $q_1()$ and $q_2()$,

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

Then

$$p(s|b, d) = \frac{q_1(s, b)q_2(b, d)}{\sum_s q_1(s, b)q_2(b, d)} = \frac{q_1(s, b)}{\sum_s q_1(s, b)}$$

which is a function of s and b but not of d. So $s \perp\!\!\!\perp d|b$. This is called the "factorisation criterion"

11 Towards data

If we know the structure of the model we can estimate CPTs or clique marginals from data:

Building CPTs from data:

```
> ## Example: Simulated data from chest network
> data(chestSim1000, package="gRbase")
> dat <- xtabs( ~., data=chestSim1000[,c("smoke","bronc","dysp")])
> ftable( dat, row.vars=1 )
      bronc yes      no
      dysp  yes  no yes  no
smoke
yes         227  49  23 166
no          133  27  45 330
```

11.1 Extracting CPTs

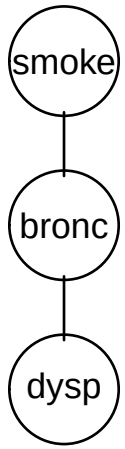
```
> ## Extract empirical distributions
> s   <- xtabs(~smoke, chestSim1000); s
smoke
yes  no
465 535
> b.s <- xtabs(~bronc+smoke, chestSim1000); b.s
     smoke
bronc yes  no
  yes 276 160
  no  189 375
> d.b <- xtabs(~dysp+bronc, chestSim1000); d.b
     bronc
dysp  yes  no
  yes 360  68
  no   76 496
```

```
> cpt.list <- compileCPT(list(s, b.s, d.b)); cpt.list
CPTspec with probabilities:
 P( smoke )
 P( bronc | smoke )
 P( dysp | bronc )
> net <- grain( cpt.list ); net
Independence network: Compiled: FALSE Propagated: FALSE
  Nodes: chr [1:3] "smoke" "bronc" "dysp"
```

11.2 Extracting clique marginals

Alternatively, we consider the undirected graph

```
> plot(ug( ~smoke:bronc+bronc:dysp ))
```



corresponding to the model

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

We might as well extract clique marginals directly:

```
> q1.sb <- xtabs(~smoke+bronc, data=chestSim1000); q1.sb
      bronc
smoke yes  no
  yes 276 189
  no  160 375
> q2.db <- xtabs(~bronc+dysp, data=chestSim1000); q2.db
      dysp
bronc yes  no
  yes 360  76
  no   68 496
```

These are clique marginals in the sense that $p(s, b) = q_1(s, b)$ and $p(b, d) = q_2(b, d)$.

It is also true that $p(b) = \sum_s q_1(s, b) = \sum_d q_2(b, d)$.

But $p(s, b, d) \neq q_1(s, b)q_2(b, d)$.

To obtain equality we must condition:

$$p(s,b,d) = p(s|b)p(b,d) = \frac{q_1(s,b)}{q_1(b)} q_2(b,d)$$

so we set $q_1(s,b) \leftarrow q_1(s,b)/q_1(s)$:

```
> q1.sb <- tableDiv(q1.sb, tableMargin(q1.sb, ~smoke)); q1.sb
      bronc
smoke   yes    no
  yes 0.594 0.406
  no  0.299 0.701
```

Now

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

and the machinery for setting evidence etc. works as before.

12 Main point

Main point: DAGs are convenient for specifying a valid joint distribution, but once this is done we do not use the DAG any more.

All computations are based on a derived underlying undirected triangulated graph.

So if we know this undirected graph we are done.

Such an undirected graph can be "learned" for example by model selection in decompsable graphical (log–linear) models for contingency tables.

Can be done, e.g. using the **gRim** package.

Once we have this graph, we have seen how to turn this structure and data into a Bayesian network.

13 Contingecy tables

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

14 Log–linear models

We are interested in modelling the *cell probabilities* p_{dhs} of a lizard falling into cell (d, h, s).

Commonly done by a hierarchical expansion of $\log p_{dhs}$ into interaction terms

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

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

If no terms are set to zero we have the *saturated model*:

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

If all interaction terms are set to zero we have the *independence model*:

$$\log p_{dhs} = \alpha^0 + \alpha^D_d + \alpha^H_h + \alpha^S_s$$

If an interaction term is set to zero then all higher order terms containing that interaction terms must also be set to zero.

For example, if we set $\beta_{dh}^{DH} = 0$ then we must also set $\gamma_{dhs}^{DHS} = 0$.

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

The non–zero interaction terms are the generators of the model. Setting $\beta_{dh}^{DH} = \gamma_{dhs}^{DHS} = 0$ the generators are

$$\{D, H, S, DS, HS\}$$

Generators contained in higher order generators can be omitted so the generators become

$$\{DS, HS\}$$

corresponding to

$$\log p_{dhs} = \alpha^{DS}_{ds} + \alpha^{HS}_{hs}$$

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

Instead of taking logs we may write p_{hds} in product form

$$p_{dhs} = q^{DS}(d, s)q^{HS}(h, s)$$

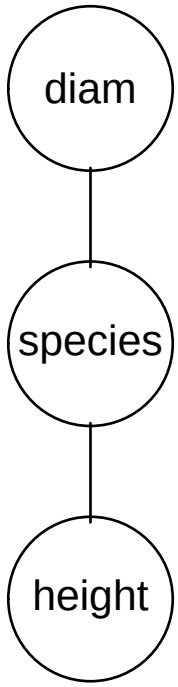
and this is in some connections useful.

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

Focus on log–linear models whose generators are the cliques of the *dependence graph*, and whose dependence graph is *triangulated*. Called decomposable log–linear graphical models:

Example: $\{DS, HS\}$ is one such models

```
> uG <- ug( ~diam:species + height:species ); plot(uG)
```



Example: $\{DS, HS, DH\}$: Not such a model; generators not cliques of dependence graph.

Example: $\{AB, BC, CD, DA\}$: Not such a model; dependence graph not triangulated.

15 Log–linear models with **gRim**

```
> ciTest(lizard, ~ height+diam+species)
Testing height _|_ diam | species
Statistic (DEV):    2.026 df: 2 p-value: 0.3632 method: CHISQ
> ciTest(lizard, ~ diam+species+height)
Testing diam _|_ species | height
Statistic (DEV):   14.024 df: 2 p-value: 0.0009 method: CHISQ
> ciTest(lizard, ~ species+height+diam)
Testing species _|_ height | diam
Statistic (DEV):   11.823 df: 2 p-value: 0.0027 method: CHISQ
```

```
> liz1 <- dmod(~.^., data=lizard)
> formula( liz1 )
~diam * height * species
> # backward search among decomposable log-linear models
> liz2 <- stepwise( liz1 )
> 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
> # turn model into Bayesian network
> grain( liz2 )
Independence network: Compiled: FALSE Propagated: FALSE
  Nodes: chr [1:3] "species" "diam" "height"
```

Coronary artery disease data:

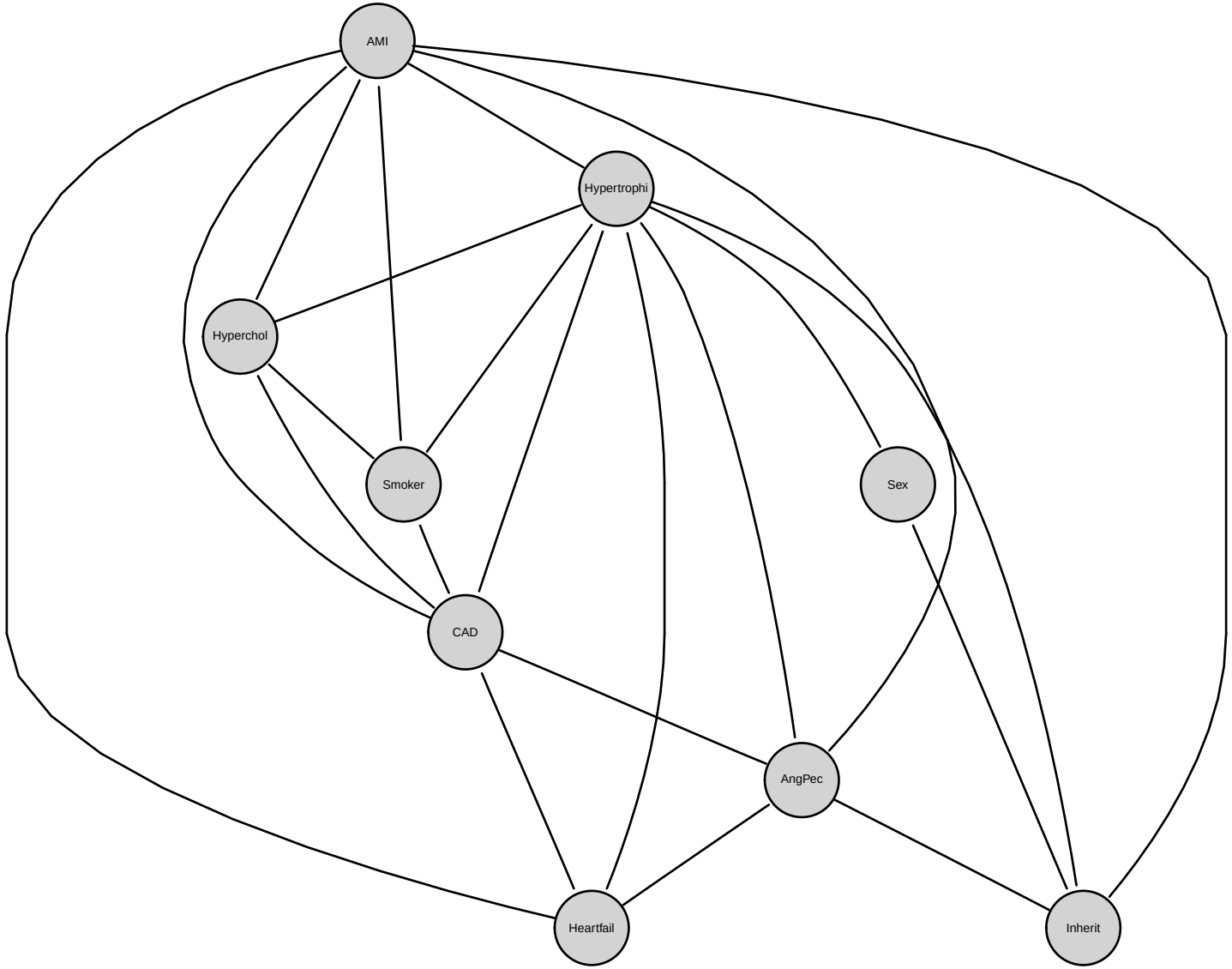
CAD is the diseae; the other variables are risk factors and disease manifestations/symptoms.

```
> 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=1, k=2 ) # use aic
STEPWISE:
 criterion: aic ( k = 2 )
 direction: backward
 type     : decomposable
 search   : all
 steps    : 1000
. BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC  -10.1543 Edge deleted: Sex CAD
  change.AIC  -10.8104 Edge deleted: Sex AngPec
  change.AIC  -18.3658 Edge deleted: AngPec Smoker
  change.AIC  -13.6019 Edge deleted: Hyperchol AngPec
  change.AIC  -10.1275 Edge deleted: Sex Heartfail
  change.AIC  -10.3829 Edge deleted: Hyperchol Heartfail
  change.AIC   -7.1000 Edge deleted: AMI Sex
  change.AIC   -9.2019 Edge deleted: Hyperchol Sex
  change.AIC   -9.0764 Edge deleted: Inherit Hyperchol
  change.AIC   -5.1589 Edge deleted: Heartfail Smoker
  change.AIC   -4.6758 Edge deleted: Inherit Heartfail
  change.AIC   -1.7378 Edge deleted: Sex Smoker
  change.AIC   -6.3261 Edge deleted: Smoker Inherit
```

```
  change.AIC   -6.2579 Edge deleted: CAD Inherit
> 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
```

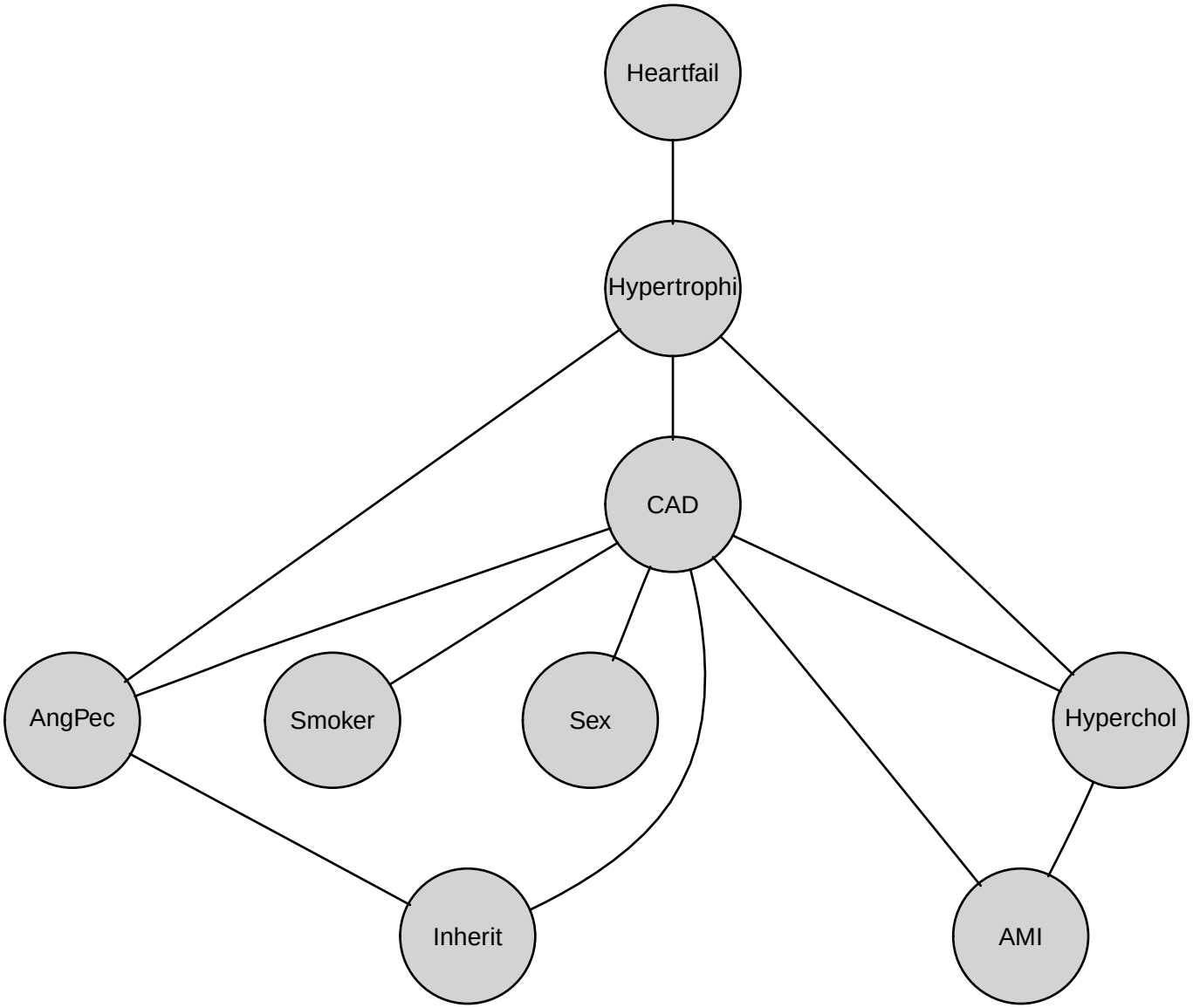
```
> plot( m.new1 )
```


AMI
Hypertrophi
Hyperchol
Smoker
Sex
CAD
AngPec
Heartfail
Inherit

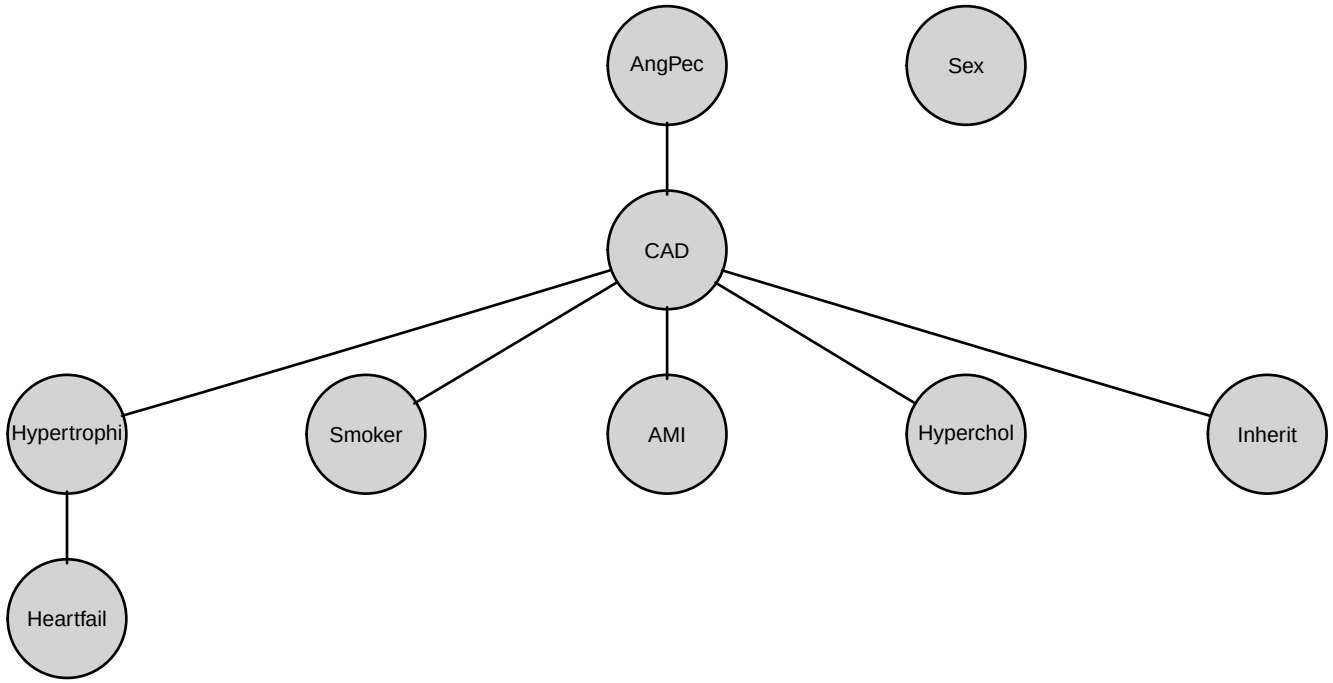
```
> m.ind <- dmod( ~.^1, data=cad1 ) # independence model
> m.new2 <- stepwise( m.ind, direction="forward", details=1, k=2 )
STEPWISE:
 criterion: aic ( k = 2 )
 direction: forward
 type     : decomposable
 search   : all
 steps    : 1000
. FORWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC  -76.5649 Edge added: AngPec CAD
  change.AIC  -64.5275 Edge added: Hypertrophi Heartfail
  change.AIC  -43.9098 Edge added: AMI CAD
  change.AIC  -32.2871 Edge added: Hyperchol CAD
  change.AIC  -17.1157 Edge added: Inherit CAD
  change.AIC  -15.8098 Edge added: CAD Hypertrophi
  change.AIC  -13.7300 Edge added: Smoker CAD
  change.AIC   -5.6746 Edge added: Sex CAD
  change.AIC   -5.3139 Edge added: AngPec Hypertrophi
  change.AIC   -1.9050 Edge added: Hypertrophi Hyperchol
  change.AIC   -1.3905 Edge added: Inherit AngPec
  change.AIC   -0.9806 Edge added: AMI Hyperchol
> 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
```

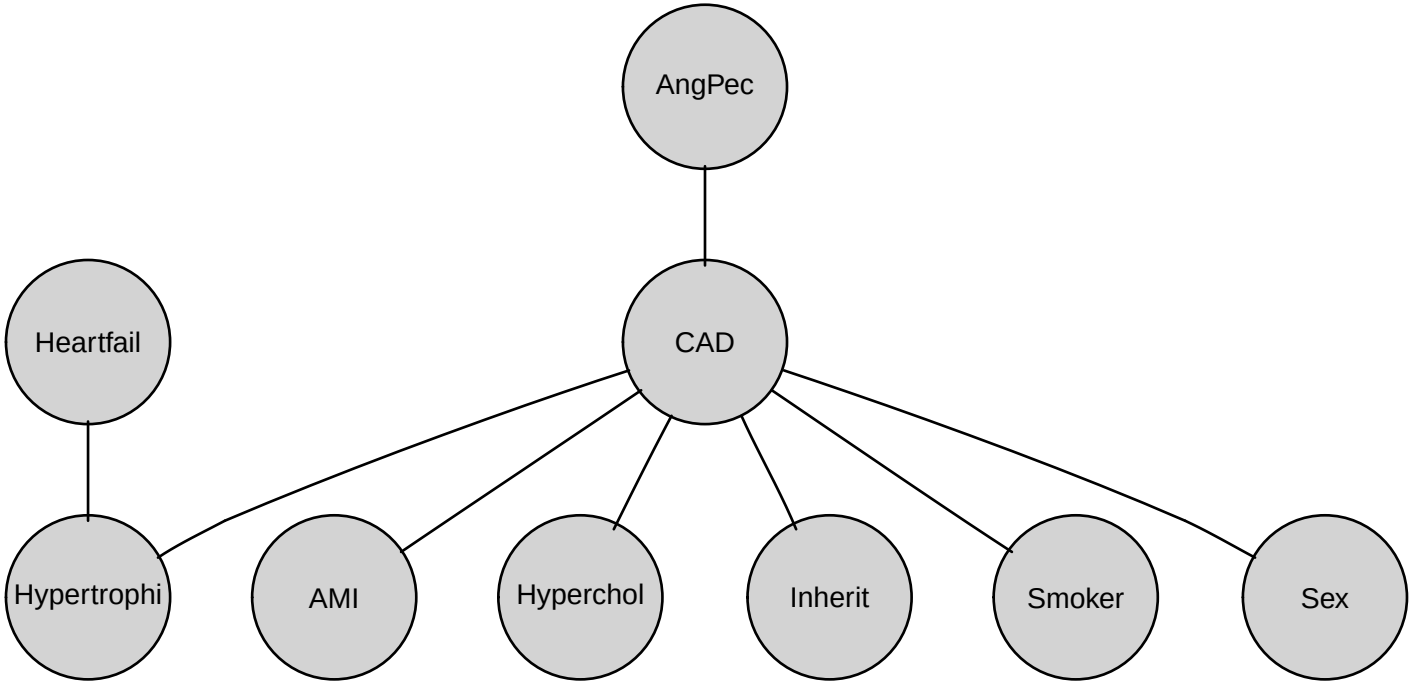
```
> plot( m.new2 )
```


Heartfail
Hypertrophi
CAD
AngPec
Smoker
Sex
Hyperchol
Inherit
AMI

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



```
> m.new4 <- stepwise( m.ind, direction="forward", k=log(nrow(cad1)) )
> plot( m.new4 )
```



16 From graph and data to network

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

```
> bn1 <- grain( m.new1, smooth=0.1 )
> bn2 <- grain( m.new2, smooth=0.1 )
> bn3 <- grain( m.new3, smooth=0.1 )
> bn4 <- grain( m.new4, smooth=0.1 )
> querygrain( bn1, "CAD")$CAD
CAD
   No   Yes
0.546 0.454
> querygrain( bn1, "CAD",
             nslist=list(AngPec="Typical", Hypertrophi="Yes"))$CAD
CAD
   No   Yes
0.599 0.401
> querygrain( bn2, "CAD")$CAD
CAD
   No   Yes
0.546 0.454
> querygrain( bn3, "CAD",
             nslist=list(AngPec="Typical", Hypertrophi="Yes"))$CAD
```

```
CAD
   No   Yes
0.503 0.497
```

17 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  9  17
```

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  9  17
> 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
```

18 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