Graphical Models and Bayesian Networks

Tutorial at useR! 2014 – Los Angeles

Søren Højsgaard

Department of Mathematical Sciences

Aalborg University, Denmark

July 1, 2014

Printed: July 1, 2014 File: bayesnet-slides.tex

Contents

1 Outline of tutorial

- 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**.

Tutorial based on these development versions:

```
> packageVersion("gRbase")
[1] '1.7.0.2'
> packageVersion("gRain")
[1] '1.2.3.1'
> packageVersion("gRim")
[1] '0.1.17.1'
```

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

Before installing the packages above, packages from bioconductor must be installed with:

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

1.2 A bit of history

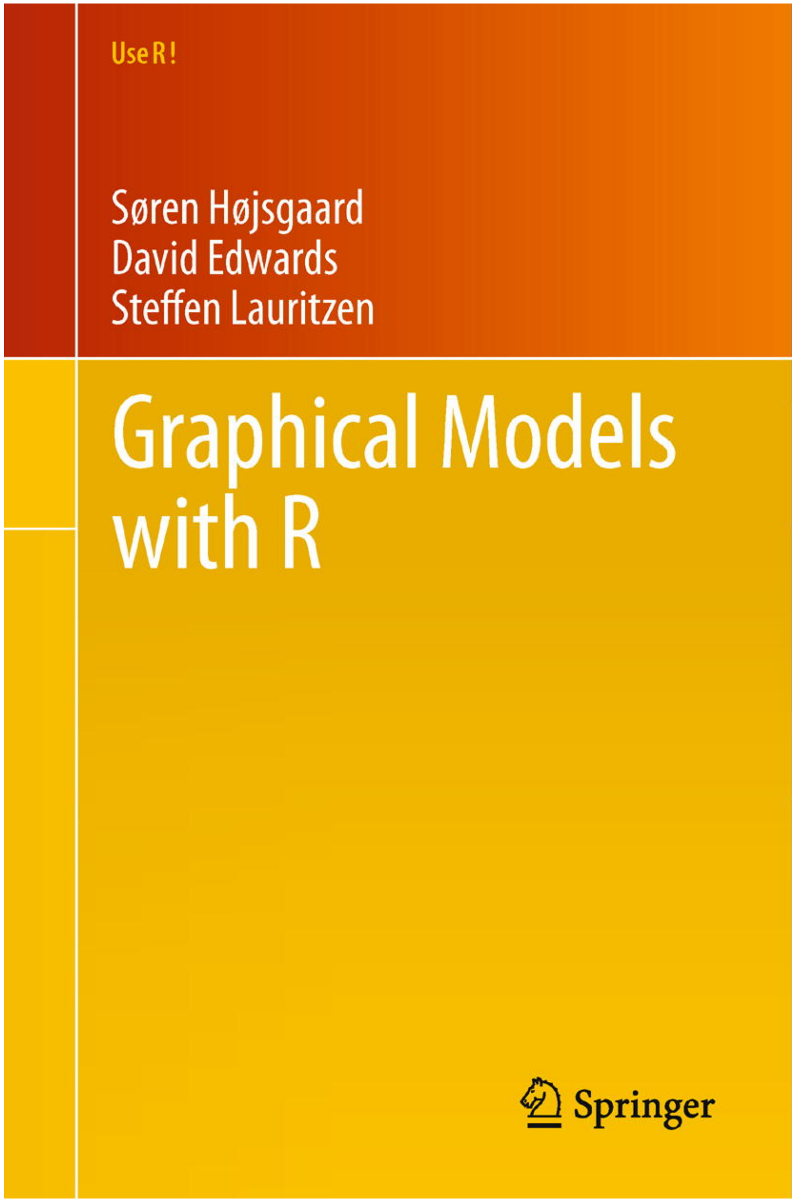
In September 2002 a small group of people gathered in Vienna for the brainstorming workshop "gR 2002" with the purpose of initiating the development of facilities in R for graphical modelling. This was made in response to the facts that:

- graphical models have now been around for a long time and have shown to have a wide range of potential applications,
- software for graphical models is currently only available in a large number of specialised packages, such as BUGS, CoCo, DIGRAM, MIM, TETRAD and others.

See also: `http://www.ci.tuwien.ac.at/gR/gR.html` and `http://www.ci.tuwien.ac.at/Conferences/gR-2002/`.

Todays workshop is one tangible result of this workshop.

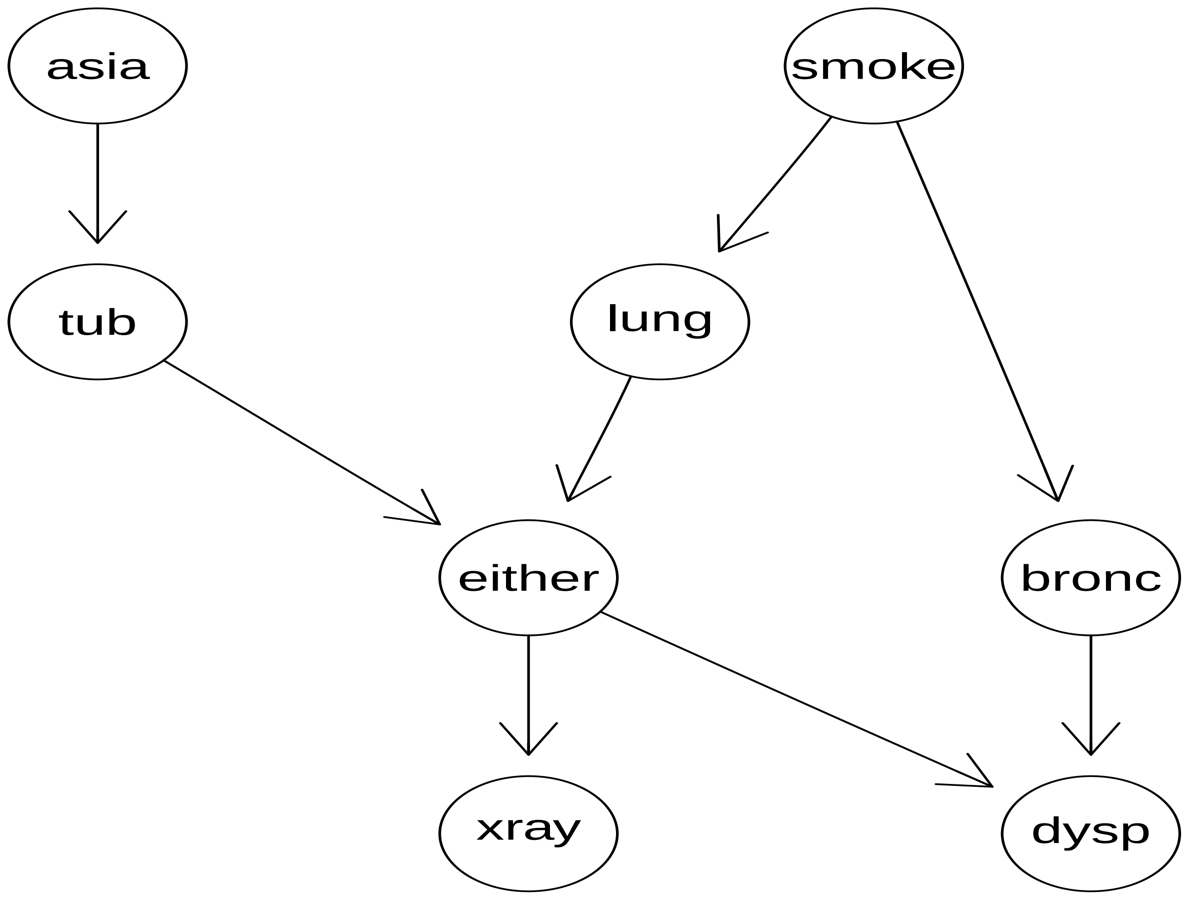
1.3 Book: Graphical Models with R



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

2 The chest clinic narrative

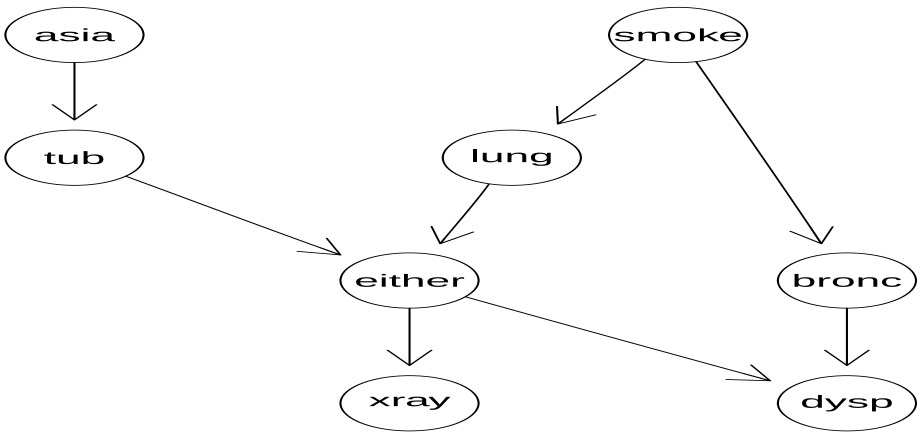


Lauritzen and Spiegehalter (1988) present the following narrative:

- "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



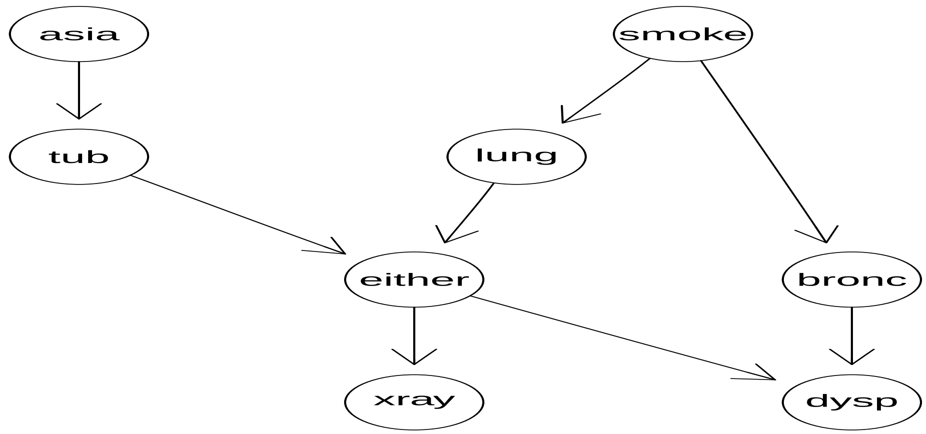
- Each node v represents a random variable Z_v
- The nodes

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

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

- We want to specify probability density

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

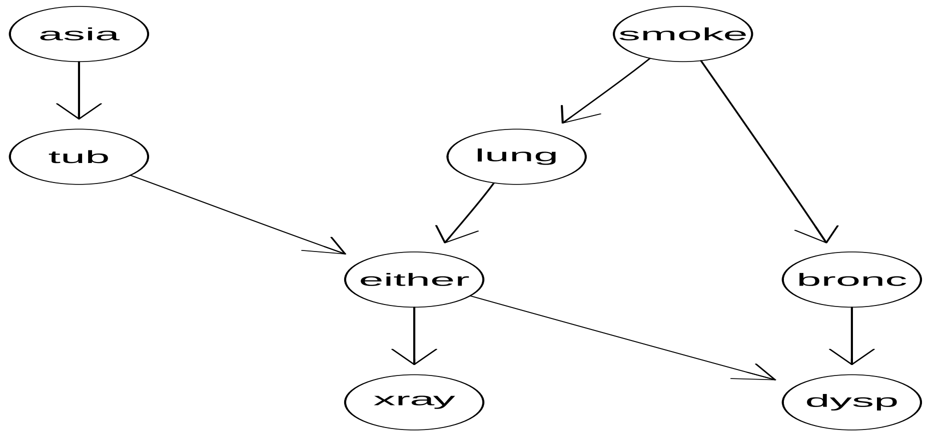


- Each node v represents a random variable Z_v (here binary with levels “yes” and “no”).

- 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

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

2.2 DAG-based models (II)

- More generally, a DAG with nodes V allows us to construct a joint distribution by combining univariate conditional distributions, i.e.

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

short for $p(z_V) = \prod_v p_{Z_v|Z_{pa(v)}}(z_v|z_{pa(v)})$.

- This is a powerful tool for constructing a multivariate distribution from univariate components.

- Example: $z_1 \sim N(a_1, \sigma_1^2)$, $z_2|z_1 \sim N(a_2 + b_2 z_1, \sigma_2^2)$, $z_3|z_2 \sim N(a_3 + b_3 z_2, \sigma_3^2)$. Then

$$p((z_1, z_2, z_3)) = p(z_1)p(z_2|z_1)p(z_3|z_2)$$

is multivariate normal

3 Conditional probability tables (CPTs)

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

```
> library(gRain)
> 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
     asia
tub   yes no
  yes   5  1
  no   95 99
```

```
> # with a formula interface
> t.a <- parray(~tub:asia, levels=list(yn,yn), 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.

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:
> bnet  <- grain(cpt.list)
> # Compile network (details follow)
> bnet <- compile(bnet)
> bnet
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(bnet, 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
> bnet.ev<-setEvidence(bnet, nodes = c("asia","dysp"),
                       states = c("yes","yes"))
> querygrain(bnet.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
```

```
> # Set additional evidence and query again
> bnet.ev<-setEvidence(bnet.ev, nodes = "xray", states = "yes")
> querygrain(bnet.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
> # Probability of observing the evidence (the normalizing constant)
> pEvidence(bnet.ev)
[1] 0.000988
```

```
> # Get joint dist of tub, lung, bronc given evidence
> x<-querygrain(bnet.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
> # Get distribution of tub given lung, bronc and evidence
> x<-querygrain(bnet.ev, nodes=c("tub","lung","bronc"),
                type="conditional")
> ftable(x, row.vars=1)
    lung     yes            no
    bronc    yes     no    yes     no
tub
yes        0.050 0.050 0.537 0.878
no         0.950 0.950 0.463 0.122
```

```
> # Remove evidence
> bnet.ev<-retractEvidence(bnet.ev, nodes="asia")
> bnet.ev
Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
  Findings: chr [1:2] "dysp" "xray"
```

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

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

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 <- cpt.list$asia
> for (i in 2:length(cpt.list)){
      joint <- tableMult( joint, cpt.list[[i]] )
  }
> dim(joint)
  dysp  bronc either   xray   lung    tub  smoke   asia
     2      2      2      2      2      2      2      2
```

```
> head( as.data.frame.table( joint ) )
  dysp bronc either xray lung tub smoke asia     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
5  yes   yes     no  yes  yes yes   yes  yes 0.00e+00
6   no   yes     no  yes  yes yes   yes  yes 0.00e+00
```

```
> ## 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
> ## equal to zero
> marg <- tableSetSliceValue(marg, c("asia","dysp"), c("yes","yes"),
                           complement=T)
> 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 $2^8 = 256$.

If there are 80 variables each with 10 levels, the joint state space is 10^{80} which is one of the estimates of 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?

7.2 So what is the solution

The trick is 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.

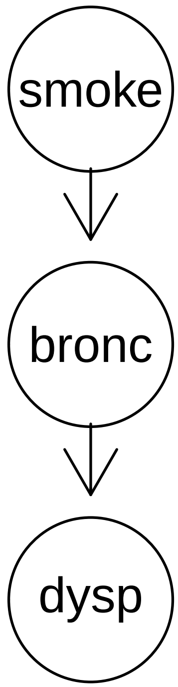
Instead we work on low dimensional tables and “send messages” between them.

With such a message passing scheme, all computations can be made locally.

The challenge is to organize these local computations.

8 Message passing – a small example

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



```
> library(gRain)
> 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))
> s; b.s; d.b
smoke
yes  no
0.5 0.5
     smoke
bronc yes no
  yes   6  3
  no    4  7
     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.

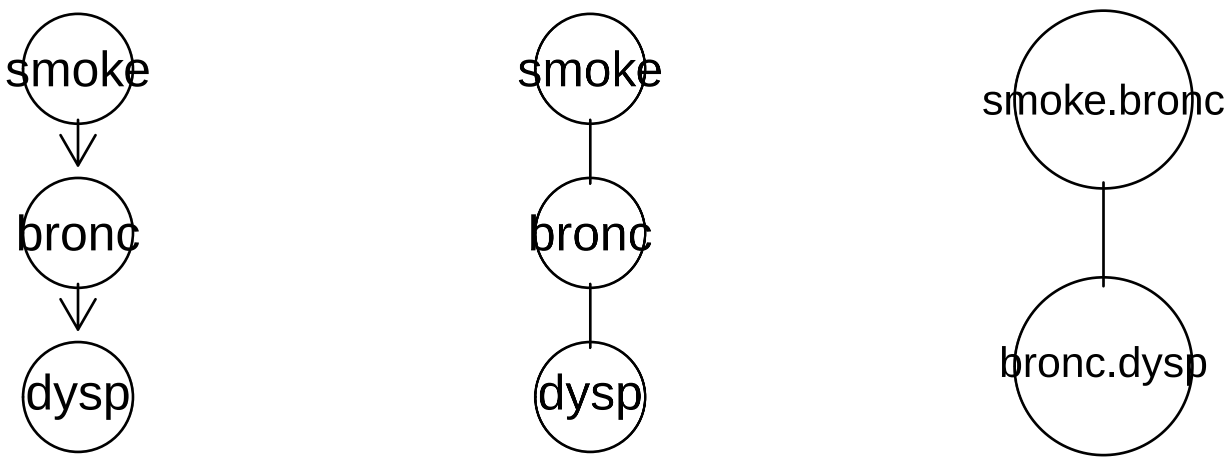
```
> joint <- tableMult( tableMult(s, b.s), d.b) ; ftable(joint)
           smoke  yes   no
dysp bronc
yes  yes         27.0 13.5
     no           4.0  7.0
no   yes          3.0  1.5
     no          16.0 28.0
```

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

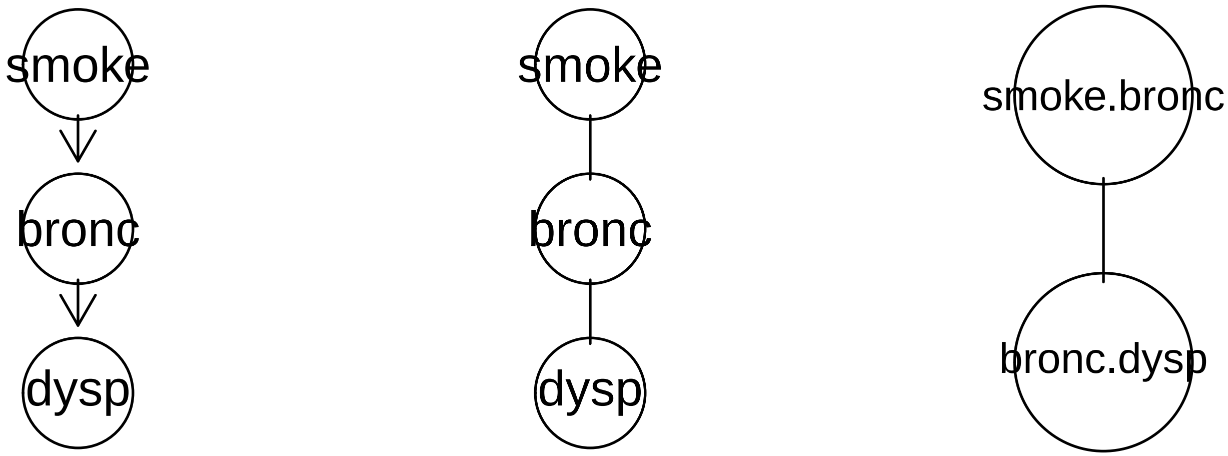
From now on we no longer need the DAG. Instead we use an undirected graph to dictate the message passing:

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.

```
> dm <-moralize(d);
> jtree<-ug(~smoke.bronc:bronc.dysp);
> par(mfrow=c(1,3)); plot(d); plot(dm); plot(jtree)
```



```
> par(mfrow=c(1,3)); plot(d); plot(dm); plot(jtree)
```



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

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

We see that 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.

We can think of the q–functions as interactions.

```
> 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
```

The factorization

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

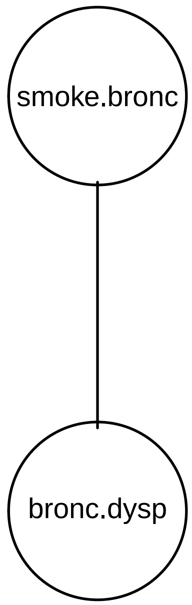
is called a clique potential representation.

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

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

8.1 Collect Evidence

```
> plot( jtree )
```



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

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

```
> q1.b  <- tableMargin(q1.sb, "bronc"); q1.b
bronc
yes  no
4.5 5.5
```

We have

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

Therefore, if 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 we obtain new potentials 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

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

is done as follows:

```
> 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.

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 set $q_1(s,b) \leftarrow q_1(s,b)q_2(b)$ and have

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

```
> 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
```

The form

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

is called the clique marginal representation and the main point is now that

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

and q_1 and q_2 "fit on their marginals", i.e. $q_1(b) = q_2(b)$

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

```
> tableMargin(q1.sb, "bronc") # or
bronc
yes  no
 45  55
> tableMargin(q2.bd, "bronc")
bronc
yes  no
 45  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 <- tableSetSliceValue(q2.bd, "dysp", "yes", complement=T); q2.
     bronc
dysp  yes no
  yes   9  2
  no    0  0
> # Repeat all the same steps as before
> 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  0
  no  11.0  0
> q1.sb <- tableDiv(q1.sb, q1.b); q1.sb
     smoke
bronc   yes    no
```

```
  yes 0.667 0.333
  no  0.364 0.636
> q2.b <- tableMargin(q2.bd, "bronc"); q2.b
bronc
 yes   no
40.5 11.0
> q1.sb <- tableMult(q1.sb, q2.b); q1.sb
     smoke
bronc yes   no
  yes  27 13.5
  no    4  7.0
```

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

```
> joint <- tableSetSliceValue(joint, "dysp", "yes", complement=T);
> ftable( joint )
           smoke  yes   no
dysp bronc
yes  yes         27.0 13.5
     no           4.0  7.0
no   yes          0.0  0.0
     no           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
```

```
> q2.bd
     dysp
bronc  yes no
  yes 40.5  0
  no  11.0  0
> tableMargin(joint, c("bronc","dysp"))
     dysp
bronc  yes no
  yes 40.5  0
  no  11.0  0
```

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):

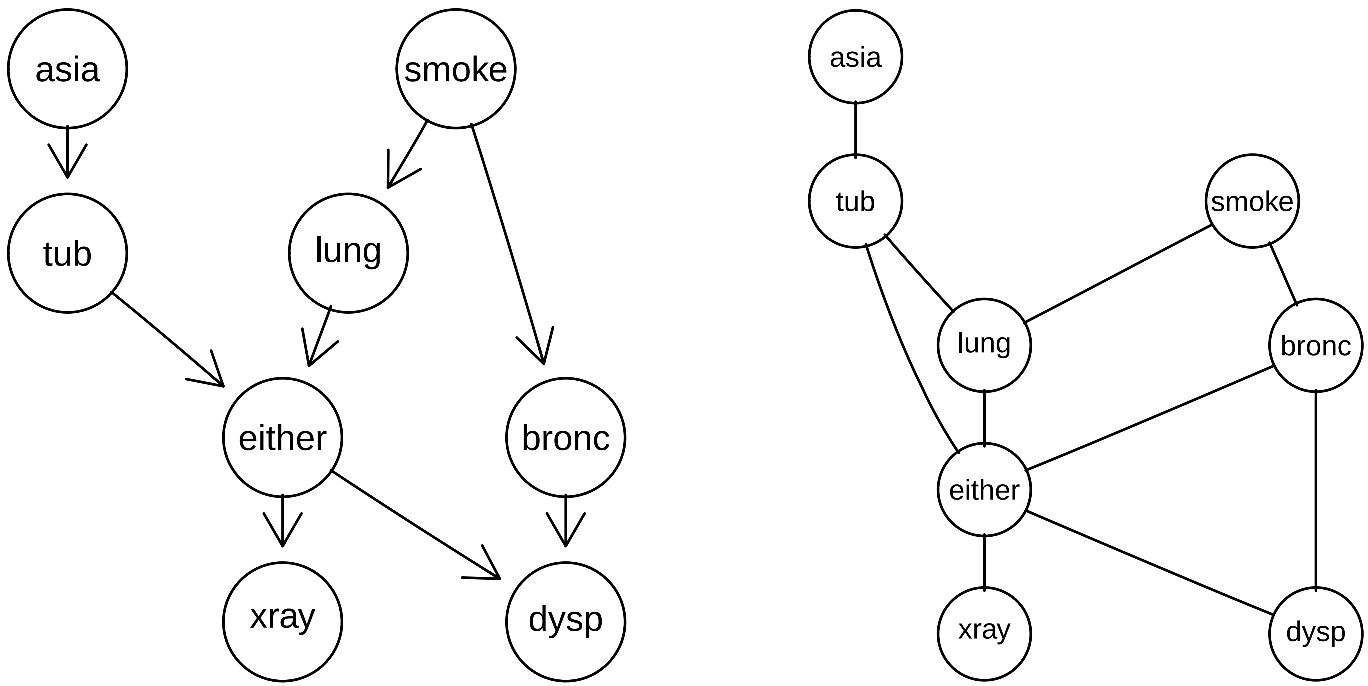
$$\begin{aligned} p(V) &= p(a)p(t|a)p(s)p(l|s)p(b|s)p(e|t,l)p(d|e,b)p(x|e) \\ &= q(a)q(t,a)q(s)q(l,s)q(b,s)q(e,t,l)q(d,e,b)q(x,e). \end{aligned}$$

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(bnet$dag); plot(moralize(bnet$dag))
```

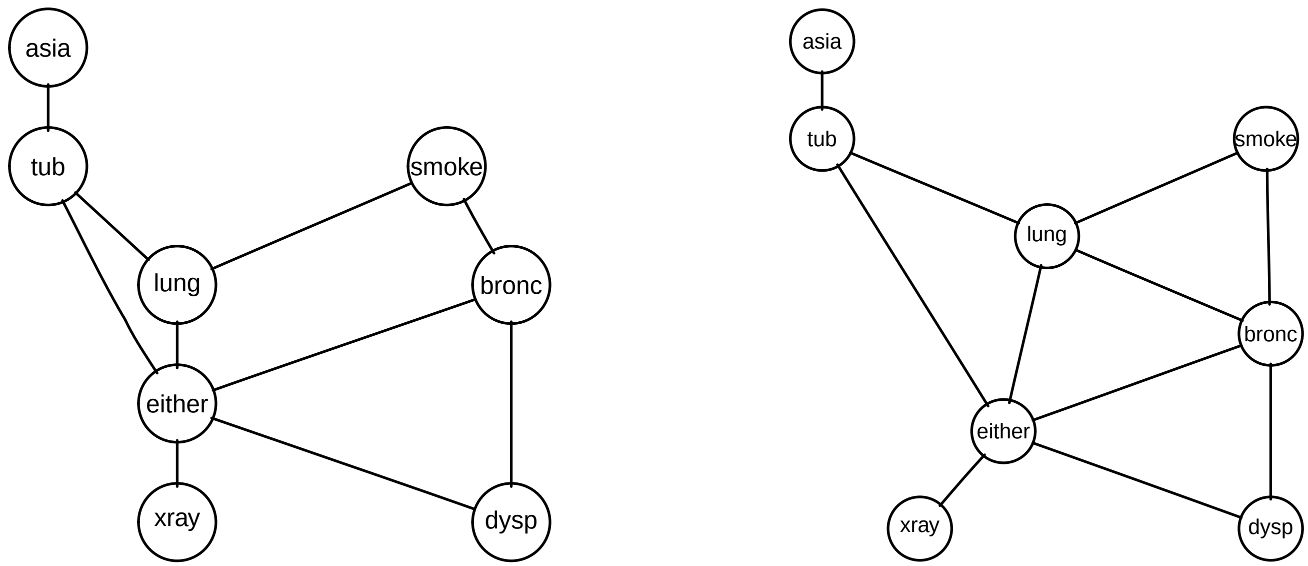


$$p(V) = q(t,a)q(l,s)q(b,s)q(e,t,l)q(d,e,b)q(x,e).$$

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(bnet$dag));
> plot(triangulate(moralize(bnet$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 $S \subset C$)

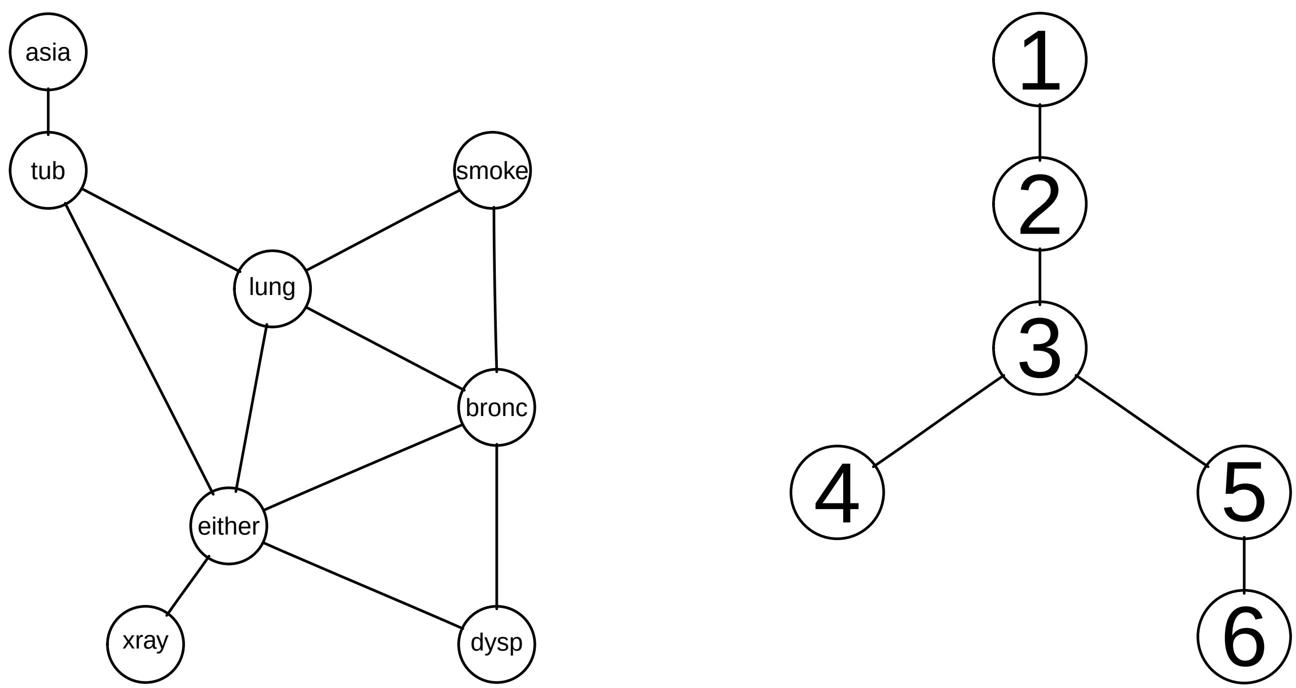
$$q(S) = \sum_{C \setminus S} q(C), \quad q(C) \leftarrow q(C)\tilde{q}(S), \quad q(C) \leftarrow q(C)/\tilde{q}(S),$$

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.

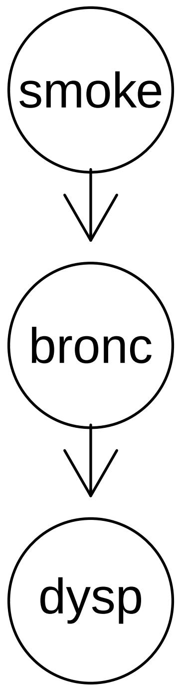
```
> par(mfrow=c(1,2)); plot(bnet$ug); plot(jTree( bnet$ug ))
> str( jTree( bnet$ug )$cliques )
List of 6
 $ : chr [1:2] "asia" "tub"
 $ : chr [1:3] "either" "lung" "tub"
 $ : chr [1:3] "either" "lung" "bronc"
 $ : chr [1:3] "smoke" "lung" "bronc"
 $ : chr [1:3] "either" "dysp" "bronc"
 $ : chr [1:2]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

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

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

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

On the other hand:

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

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 $q_2()$,

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

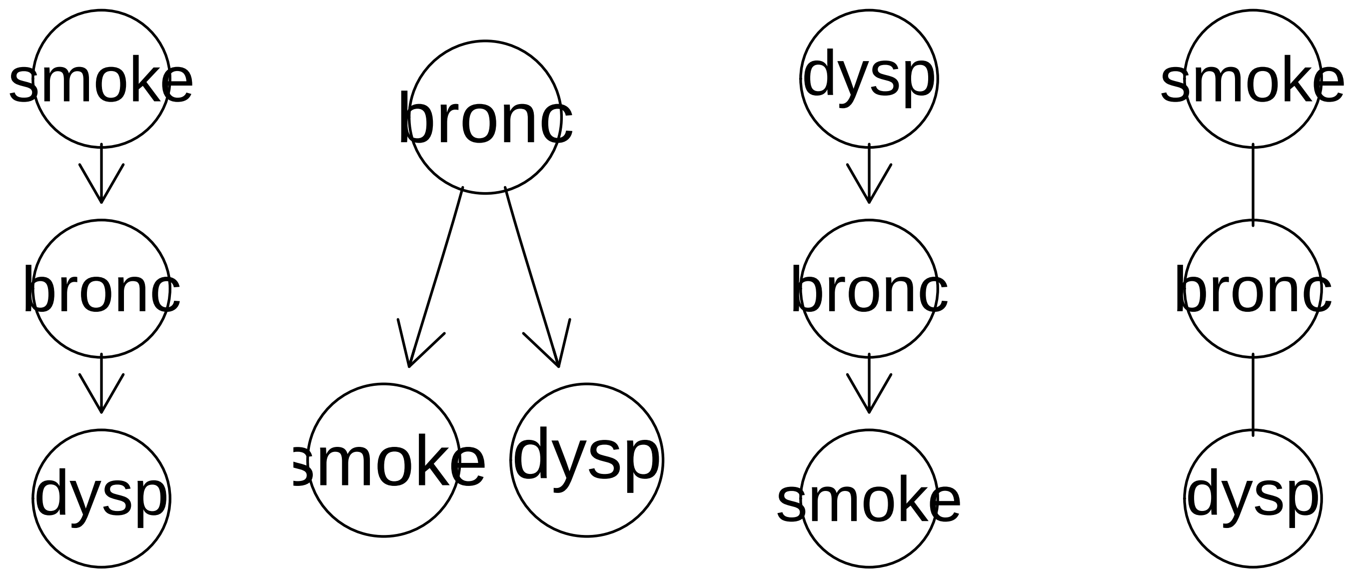
Then

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

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

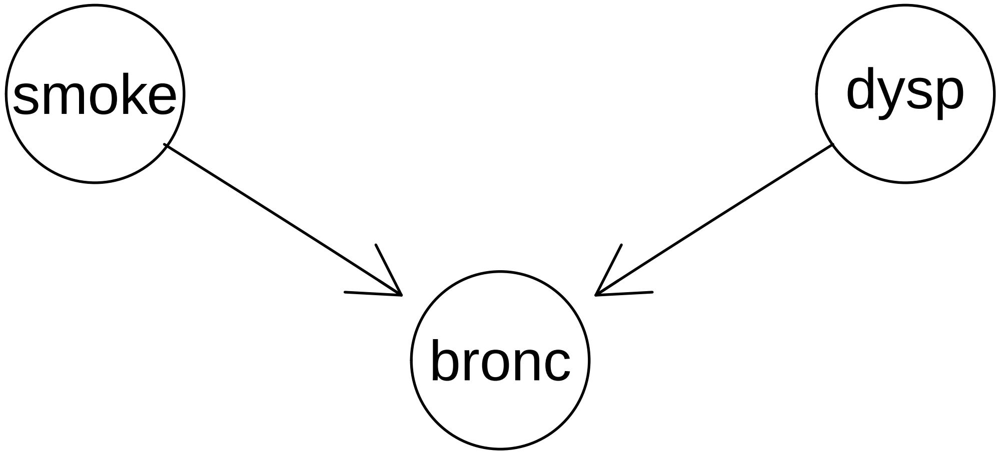
Clear that $s \perp\!\!\!\perp d|b$ under all these models:

```
> par(mfrow = c(1,4))
> plot(dag(~smoke+bronc|smoke+dysp|bronc))
> plot(dag(~bronc+smoke|bronc+dysp|bronc))
> plot(dag(~dysp+smoke|bronc+bronc|dysp))
> plot(ug(~smoke:bronc+bronc:dysp))
```



The general "rule" is therefore that separation in a graph corresponds to conditional independence – but there is an exception

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



corresponding to

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

No factorization – and no conditional independence.

11 Towards data

Building CPTs from data:

```
> ## Example: Simulated data from chest network
> 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
```

11.1 Extracting CPTs

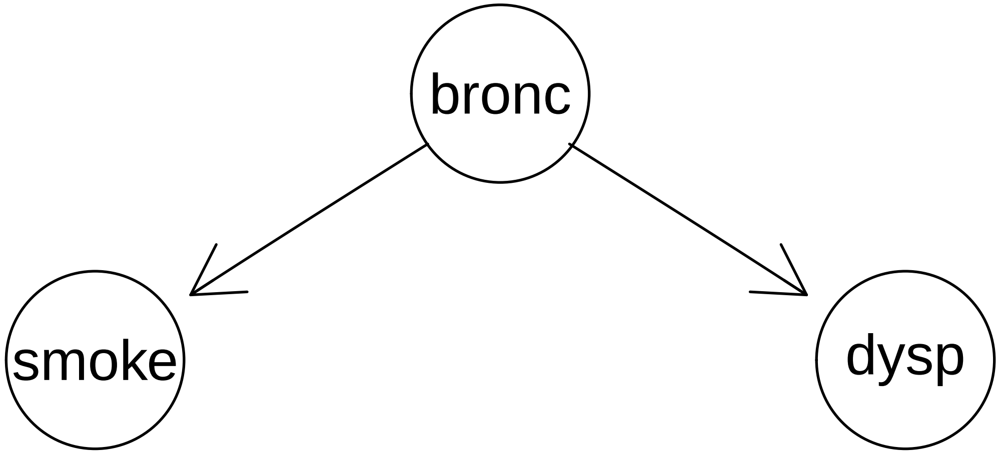
```
> ## Extract empirical distributions
> s   <- xtabs(~smoke, chestSim1000); s
smoke
yes  no
465 535
> b.s <- xtabs(~bronc+smoke, chestSim1000); b.s
     smoke
bronc yes  no
  yes 276 160
  no  189 375
> d.b <- xtabs(~dysp+bronc, chestSim1000); d.b
     bronc
dysp  yes  no
  yes 360  68
  no   76 496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

```
> ## Normalize to CPTs if desired (not necessary because
> ## we can always normalize at the end)
> s   <- as.parray(s, normalize="first"); s
smoke
  yes    no
0.465 0.535
> b.s <- as.parray(b.s, normalize="first"); b.s
     smoke
bronc   yes    no
  yes 0.594 0.299
  no  0.406 0.701
> d.b <- as.parray(d.b, normalize="first"); d.b
     bronc
dysp    yes    no
  yes 0.826 0.121
  no  0.174 0.879
```

```
> 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"
```

But we could just as well extract CPTs for this model,

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

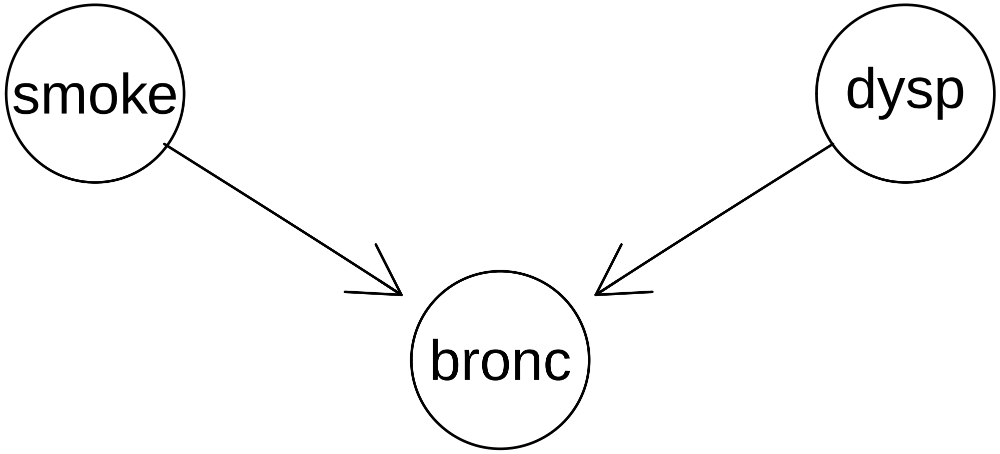


in the sense that the joint distribution will become the same:

```
> ## Extract empirical distributions
> b   <- xtabs(~bronc, chestSim1000);
> s.b <- xtabs(~smoke+bronc, chestSim1000);
> d.b <- xtabs(~dysp+bronc, chestSim1000);
```

Notice, that in this case

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



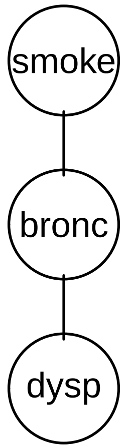
the joint distribution will be different:

```
> ## Extract empirical distributions
> s    <- xtabs(~smoke, chestSim1000);
> d    <- xtabs(~dysp, chestSim1000);
> b.sd <- xtabs(~bronc+smoke+dysp, chestSim1000);
```

11.2 Extracting clique marginals

Alternatively, we consider the undirected graph

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



corresponding to the model

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

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 $p(b, d) = q_2(b, d)$. Hence $p(s, b, d) \neq q_1(s, b)q_2(b, d)$. But it is true that $p(b) = \sum_s q_1(s, b) = \sum_d q_2(b, d)$.

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

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

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

12 Learning the model structure

The next step is to "learn" the structure of association between the variables.

By this we mean learn the conditional independencies among the variables from data.

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

12.1 Contingency tables

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

```
> data(lizardRAW, package="gRbase")
> dim(lizardRAW)
[1] 409   3
> 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
```

Let $V = \{D, H, S\}$. We have 409 observations of *discrete random vectors* $Z = Z_V = (Z_D, Z_H, Z_S S)$ where each component is binary.

A *configuration* of Z is denoted by $z = (z_D = d, z_H = h, z_S = s)$ (which we shall also write as (d, h, s)).

It is common to organize such data in a *contingency table*

```
> lizard<-xtabs(~., data=lizardRAW)
> dim( lizard )
[1] 2 2 2
> ftable( lizard )
            species anoli dist
diam height
<=4  <=4.75            86   73
     >4.75             32   61
>4   <=4.75            35   70
     >4.75             11   41
```

A configuration z is also a *cell* in a contingency table. The *counts* in cell z is denoted by $n(z)$ or by $n(d, h, s)$.

The probability of a configuration $z = (d, h, s)$ is denoted $p(z)$ and this is also the probability of a lizard falling in the (d, h, s) cell.

One estimate of the probabilities is by the relative frquencies:

```
> lizardProb <- lizard/sum(lizard); ftable(lizardProb)
            species  anoli   dist
diam height
<=4  <=4.75         0.2103 0.1785
     >4.75          0.0782 0.1491
>4   <=4.75         0.0856 0.1711
     >4.75          0.0269 0.1002
```

For $A \subset V$ we have a marginal table with counts $n(z_A)$, for example

```
> tableMargin(lizard, ~height+species)
        species
height   anoli dist
  <=4.75   121  143
  >4.75     43  102
```

The probability of an observation in a marginal cell z_A is $p(z_A) = \sum_{z':z'_A=z_A} p(z')$. For example

```
> tableMargin(lizardProb, ~height+species)
        species
height   anoli  dist
  <=4.75 0.296 0.350
  >4.75  0.105 0.249
```

12.2 Log–linear models

We are interested in modelling the <u>*cell probabilities*</u> p_{dhs}.

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 <u>*saturated model*</u>:

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

If all interaction terms are set to zero we have the <u>*independence model*</u>:

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

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 $D \perp\!\!\!\perp H \mid S$.

In the context of these data, $D \perp\!\!\!\perp H \mid S$ means there there is independence between D and H in each slice defined by species S.

Just looking at data, this looks reasonable.

```
> 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
```

12.3 Hierarchical log–linear models

More generally the *generating class* of a log–linear model is a set $\mathcal{A} = \{A_1, \ldots, A_Q\}$ where $A_q \subset V$.

This corresponds to

$$p(z) = \prod_{A \in \mathcal{A}} q_A(z_A)$$

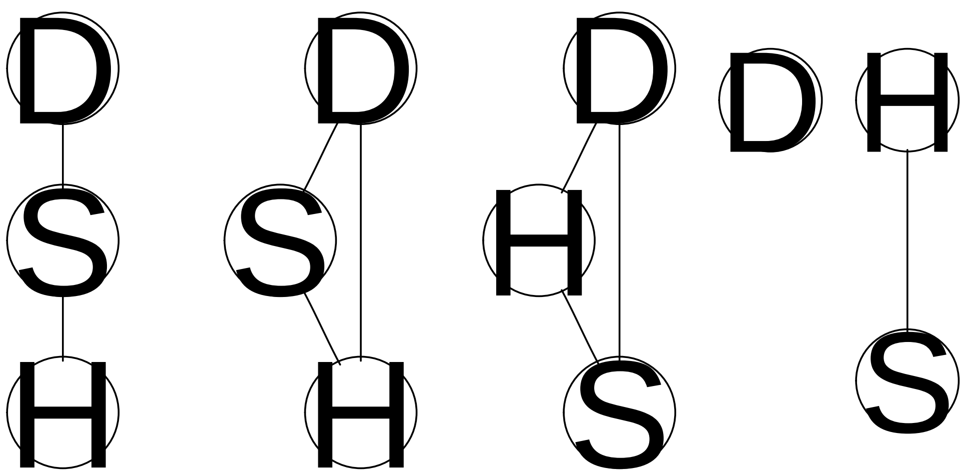
where q_A is a potential, a function that depends on z only through z_A.

12.4 Dependence graphs

The *dependence graph* for the model has nodes V and undirected edges E given as follows: $\{v_1, v_2\}$ is in E iff $\{v_1, v_2\} \subset A_q$ for some $A_q \in \mathcal{A}$.

Example: $\{DS, HS\}$, $\{DS, HS, DH\}$, $\{DHS\}$, $\{D, HS\}$ have these dependence graphs:

```
> par(mfrow=c(1,4))
> plot( ug(~D:S + H:S ))
> plot( ug(~D:S + H:S + D:H ))
> plot( ug(~D:H:S ))
> plot( ug(~D + H:S ))
```

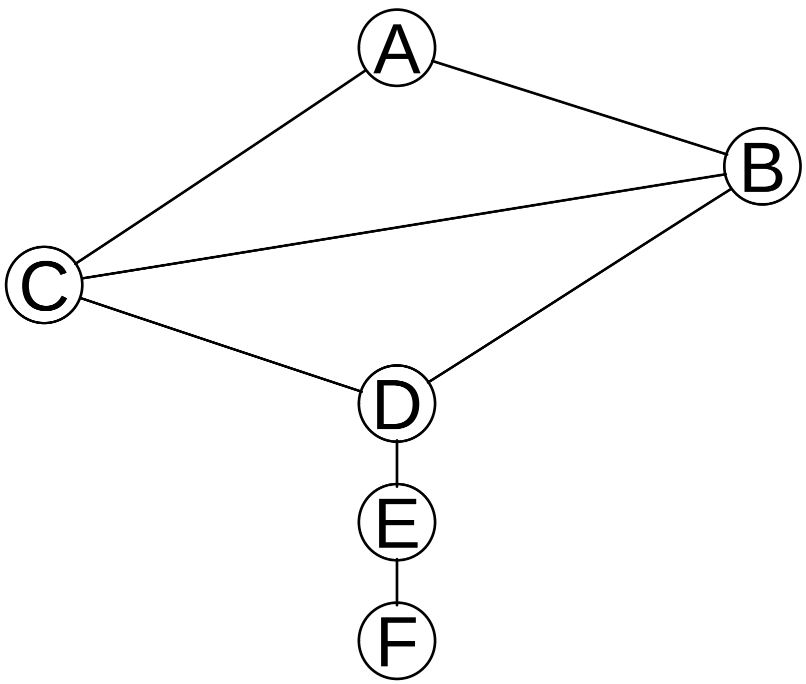


12.5 The Global Markov property

There is a general rule reading conditional independencies from a graph: If two sets of nodes U and V are separated by a third set W then $U \perp\!\!\!\perp V | W$.

Example: $\{E, F\} \perp\!\!\!\perp A | \{B, C\}$.

```
> plot( ug(~A:B:C+B:C:D+D:E+E:F ))
```



12.6 Estimation – likelihood equations

Under *multinomial sampling* the likelihood is

$$L = \prod_{\text{all states } z} p(z)^{n(z)} = \prod_{A \in \mathcal{A}} \prod_{z_A} q_A(z_A)^{n(z_A)}$$

The MLE $\hat{p}(z)$ for $p(z)$ is the (unique) solution to the likelihood equations

$$\hat{p}(z_A) = n(z_A)/n, \quad A \in \mathcal{A}$$

Typically MLE must be found by iterative methods, e.g. iterative proportional scaling (IPS).

However, for some log–linear models (called decomposable models) the MLE can be found in closed form. In this case IPS converges in 2 iterations.

12.7 Fitting log–linear models

Iterative proportional scaling is implemented in *loglin()*:

```
> ll1 <- loglin(lizard, list(c("species","diam"),
                              c("species","height")))
2 iterations: deviation 0
> str( ll1 )
List of 4
 $ lrt    : num 2.03
 $ pearson: num 2.02
 $ df     : num 2
 $ margin :List of 2
  ..$ : chr [1:2] "species" "diam"
  ..$ : chr [1:2] "species" "height"
```

A formula based interface to *loglin()* is provided by *loglm()*:

```
> library(MASS)
> ll2 <- loglm(~species:diam + species:height, data=lizard); ll2
Call:
loglm(formula = ~species:diam + species:height, data = lizard)

Statistics:
                  X^2 df P(> X^2)
Likelihood Ratio 2.03  2    0.363
Pearson          2.02  2    0.365
> coef( ll2 )
$`(Intercept)`
[1] 3.79

$diam
   <=4      >4
 0.283 -0.283

$height
<=4.75  >4.75
 0.343 -0.343
```

```
$species
 anoli   dist
-0.309  0.309

$diam.species
     species
diam   anoli   dist
  <=4  0.188 -0.188
  >4  -0.188  0.188

$height.species
        species
height     anoli   dist
  <=4.75  0.174 -0.174
  >4.75  -0.174  0.174
```

The *dmod()* function also provides an interface to *loglin()*, and *dmod()* offers much more; see later.

```
> library(gRim)
> ll3 <- dmod(~species:diam + species:height, data=lizard); ll3
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
```

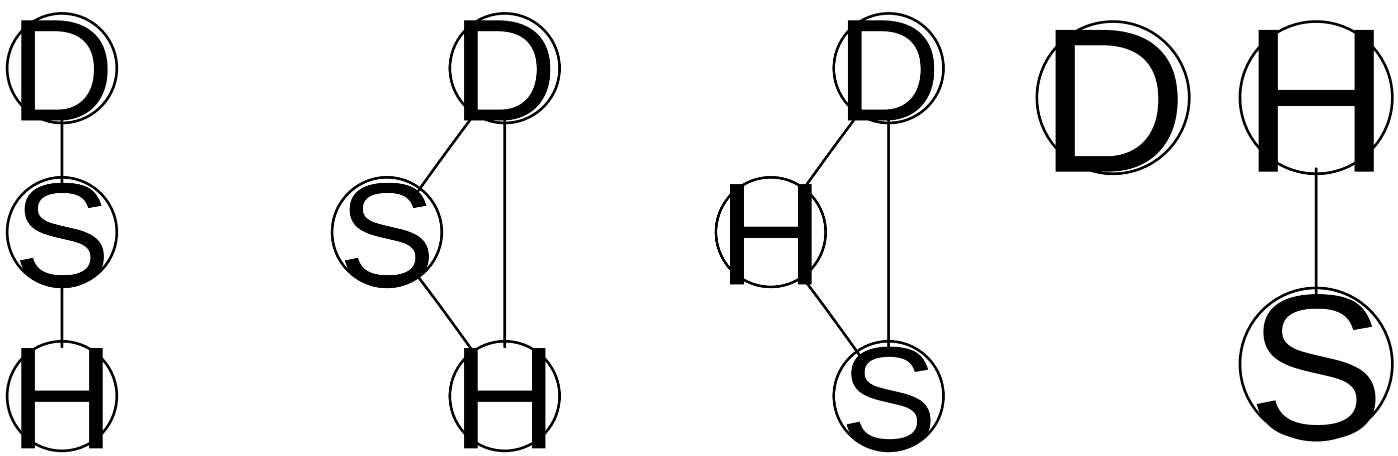
12.8 Graphical models and decomposable models

Let $Z = (Z_v, v \in V)$ be a random vector and let $\mathcal{A} = \{A_1, \ldots, A_Q\}$ where $A_q \subset V$ be a generating class for a log linear model corresponding to

$$p(z) = \prod_{A \in \mathcal{A}} q_A(z_A)$$

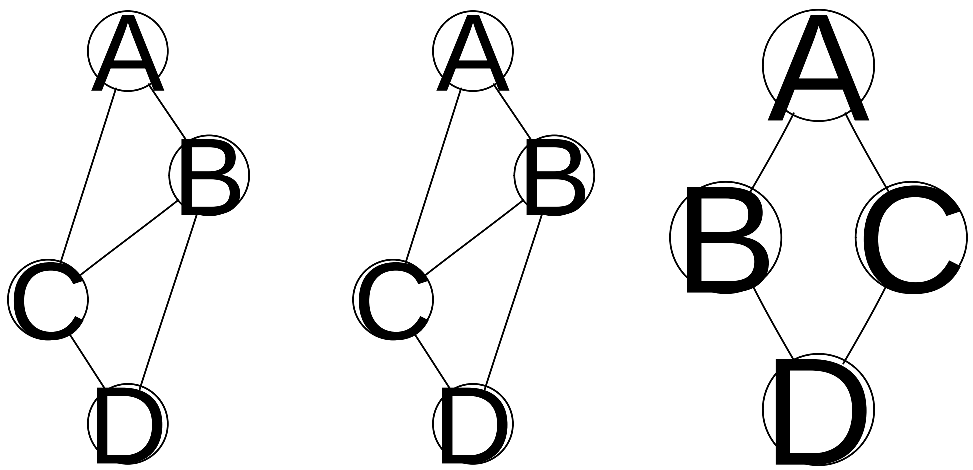
Definition 1 *A hierarchical log–linear model with generating class* $\mathcal{A} = \{a_1, \dots a_Q\}$ *is* graphical *if* $\mathcal{A}$ *are the cliques of the dependence graph.*

```
> par(mfrow=c(1,4))
> plot( ug(~D:S + H:S ))        ## graphical
> plot( ug(~D:S + H:S + D:H )) ## not graphical
> plot( ug(~D:H:S ))            ## graphical
> plot( ug(~D + H:S ))          ## graphical
```



Definition 2 *A graphical log–linear model is* decomposable *if its dependence graph is triangulated (has no* $\geq$ *4–cycles). Only graphical models can be decomposable.*

```
> par(mfrow=c(1,3))
> plot(ug(~A:B:C + B:C:D))            ## graphical, decomposable
> plot(ug(~A:B + A:C + B:C:D))        ## not graphical, not decomposable
> plot(ug(~A:B + A:C + B:D + C:D)) ## graphical, not decomposable
```



12.9 ML estimation in decomposable models

Major point: ML estimates in decomposable models can be found in closed form (no iterations). Consider lizard data:

The saturated model $\{DHS\}$ (i.e. no restrictions on p_{dhs}) is decomposable, and the MLE is

$$\hat{p}_{dhs} = n(d,h,s)/n$$

Next consider the decomposable model $\{DS, HS\}$. The term interaction DS can also be seen as the saturated model for the marginal table

```
> n.ds <- tableMargin(lizard, ~diam+species); n.ds
     species
diam  anoli dist
  <=4   118  134
  >4     46  111
```

i.e. there is no restriction on p_{ds}, and the MLE is $\hat{p}_{ds} = n(d,s)/n$.

Generally, for a decomposable model, the MLE can be found in closed form as

$$\widehat{p}(z) = \frac{\prod_{C:cliques} \widehat{p}_C(z_C)}{\prod_{S:separators} \widehat{p}_S(z_S)}$$

where $\widehat{p}_E(z_E) = n(z_E)/n$ for any clique or separator E.

So for $\{DS, HS\}$ we have

$$\widehat{p}_{dhs} = \frac{\widehat{p}_{ds}\widehat{p}_{hs}}{\widehat{p}_s} = \frac{[n(d,s)/n][n(h,s)/n]}{n(s)/n}$$

It is easy to see that we have the MLE: The MLE $\widehat{p}_{dhs}$ is the solution to the equation

$$\widehat{p}_{ds} = n(d,s)/n, \quad \widehat{p}_{hs} = n(h,s)/n$$

```
> n.ds <- tableMargin(lizard, c("diam",   "species"))
> n.hs <- tableMargin(lizard, c("height", "species"))
> n.s  <- tableMargin(lizard, c("species"))
> ec   <- tableDiv( tableMult(n.ds, n.hs), n.s) ## expected counts
> ftable( ec )
               diam  <=4   >4
species height
anoli   <=4.75      87.1 33.9
        >4.75       30.9 12.1
dist    <=4.75      78.2 64.8
        >4.75       55.8 46.2
> ftable( fitted(ll2) )
Re-fitting to get fitted values
            species anoli dist
diam height
<=4  <=4.75          87.1 78.2
     >4.75           30.9 55.8
>4   <=4.75          33.9 64.8
     >4.75           12.1 46.2
```

13 Decomposable models and Bayesian networks

Now is the time to establish connections between decomposable graphical models and Bayesian networks.

- For a decomposable model, the MLE is given as

$$\hat{p}(z) = \frac{\prod_{C:cliques} \hat{p}_C(z_C)}{\prod_{S:separators} \hat{p}_S(z_S)} = \frac{\prod_{C:cliques} n(z_C)/n}{\prod_{S:separators} n(z_S)/n}$$

- Major point: The above is IMPORTANT in connection with Bayesian networks, it is a *clique potential* representation of p.
- Hence if we find a decomposable graphical model then we can convert this to a Bayesian network.
- We need not specify conditional probability tables (they are only used for specifying the model anyway, the real computations takes place in the junction tree).

- There are $2^{K_{n,2}}$ graphical models with n variables, so model search is a challenge. The number of decomposable models is smaller and these models can be fitted without iterations so model search among decomposable models is faster.

14 Testing for conditional independence

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

```
> library(gRim)
> args(ciTest_table)
function (x, set = NULL, statistic = "dev", method = "chisq",
    adjust.df = TRUE, slice.info = TRUE, L = 20, B = 200, ...)
NULL
```

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
```

14.1 What is a CI-test – stratification

Conditional independence of u and v given W means independence of u and v for each configuration w^* of W.

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

Conceptually form a factor S by crossing the factors in W. The test can then be formulated as a test of the conditional independence $u \perp\!\!\!\perp v \mid S$ in a three way table.

The deviance decomposes into independent contributions from each stratum:

$$D = 2 \sum_{ijs} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_s 2 \sum_{ij} n_{ijs} \log \frac{n_{ijs}}{\hat{m}_{ijs}} = \sum_s D_s$$

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

```
> 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 $\{n_{ijs}\}_{i=1\dots|u|,j=1\dots|v|}$. 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.

14.2 Example: University admissions

Example: Admission to graduate school at UC at Berkley in 1973 for the six largest departments classified by sex and gender.

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

Is there evidence of sexual discrimination?

```
> ag <- tableMargin(UCBAdmissions, ~Admit+Gender); ag
          Gender
Admit       Male Female
  Admitted 1198    557
  Rejected 1493   1278
> as.parray( ag, normalize="first" )
          Gender
Admit        Male Female
  Admitted 0.445  0.304
  Rejected 0.555  0.696
```

```
> s<-ciTest(UCBAdmissions, ~Admit+Gender+Dept, slice.info=T); s
Testing Admit _|_ Gender | Dept
Statistic (DEV):   21.736 df: 6 p-value: 0.0014 method: CHISQ
```

Hence, `admit` and `gender` are not independent within each `Dept`.

However, most contribution to the deviance comes from department `A`:

```
> s$slice
  statistic  p.value df Dept
1    19.054 1.27e-05  1    A
2     0.259 6.11e-01  1    B
3     0.751 3.86e-01  1    C
4     0.298 5.85e-01  1    D
5     0.990 3.20e-01  1    E
6     0.384 5.36e-01  1    F
```

So what happens in department `A`?

```
> x <- tableSlice(UCBAdmissions, margin="Dept", level="A"); x
          Gender
Admit      Male Female
  Admitted  512     89
  Rejected  313     19
> as.parray(x, normalize="first")
          Gender
Admit       Male Female
  Admitted 0.621  0.824
  Rejected 0.379  0.176
```

The discrimination is against men!

Why were we mislead at the beginning?

```
> x <- tableMargin(UCBAdmissions, ~Admit+Dept);
> x
          Dept
Admit        A   B   C   D   E   F
  Admitted 601 370 322 269 147  46
  Rejected 332 215 596 523 437 668
> as.parray(x, norm="first")
          Dept
Admit          A     B     C    D     E      F
  Admitted 0.644 0.632 0.351 0.34 0.252 0.0644
  Rejected 0.356 0.368 0.649 0.66 0.748 0.9356
```

15 Log–linear models – the **gRim** package

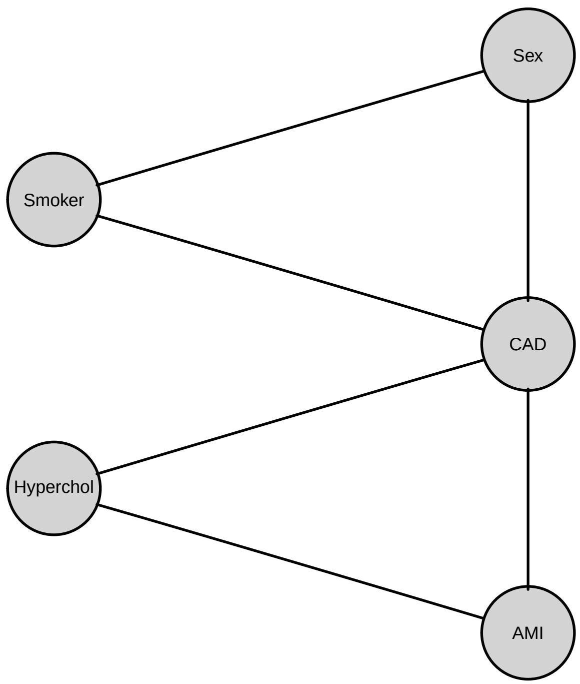
Coronary artery disease data:

```
> 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
```

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

Some (random) model:

```
> m1 <- dmod(~Sex:Smoker:CAD + CAD:Hyperchol:AMI, data=cad1); m1
Model: A dModel with 5 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :        1293.88 mdim :   13 aic :       1319.88
 ideviance :         112.54 idf  :    8 bic :       1364.91
 deviance  :          16.38 df   :   18
> plot( m1 )
```



- Data must be a table or a dataframe (which will be converted to a table).
- Variable names may be abbreviated.
- Instead of a formula, a list can be given.
- The *generating class* as a list is retrieved with *terms()* and as a formula with *formula()*:

```
> str( terms( m1 ) )
List of 2
 $ : chr [1:3] "Sex" "Smoker" "CAD"
 $ : chr [1:3] "CAD" "Hyperchol" "AMI"
> formula( m1 )
~Sex * Smoker * CAD + CAD * Hyperchol * AMI
```

Notice: No dependence graph in model object; must be generated on the fly using *ugList()*:

```
> # Default: a graphNEL object
> DG <- ugList( terms( m1 ) ); DG
A graphNEL graph with undirected edges
Number of Nodes = 5
Number of Edges = 6
> # Alternative: an adjacency matrix
> a <- ugList( terms( m1 ), result="matrix" ); a
          Sex Smoker CAD Hyperchol AMI
Sex         0      1   1         0   0
Smoker      1      0   1         0   0
CAD         1      1   0         1   1
Hyperchol   0      0   1         0   1
AMI         0      0   1         1   0
> A <- ugList( terms( m1 ), result="dgCMatrix" )
```

15.1 Model specification shortcuts

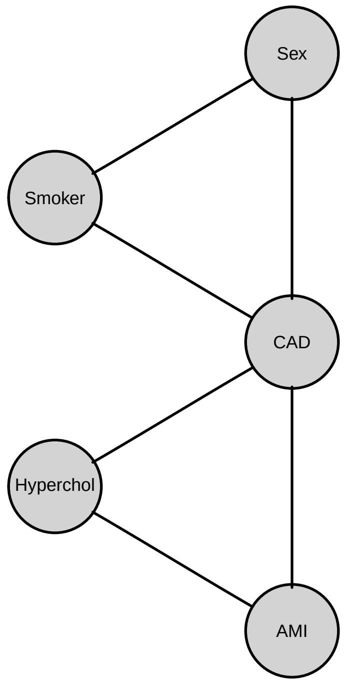
Shortcuts for specifying some models

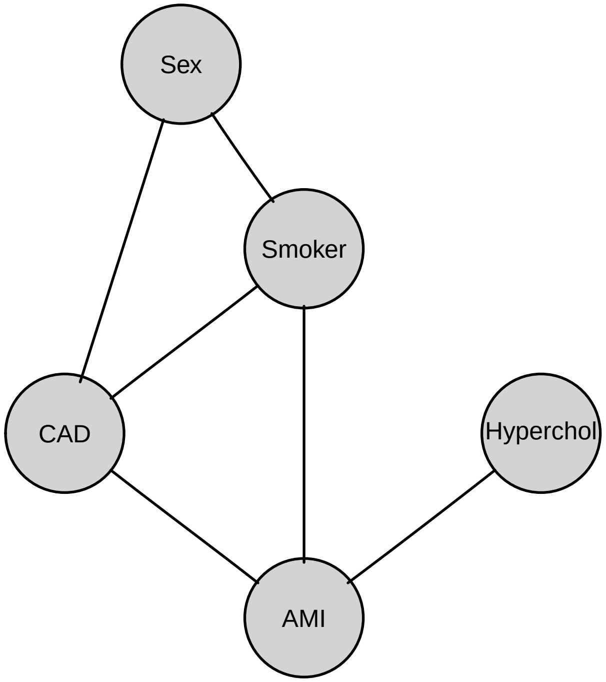
```
> mar <- c("Sex","AngPec","AMI","CAD")
> str(terms(dmod(~.^., data=cad1, margin=mar))) ## Saturated model
List of 1
 $ : chr [1:4] "Sex" "AngPec" "AMI" "CAD"
> str(terms(dmod(~.^1, data=cad1, margin=mar))) ## Independence model
List of 4
 $ : chr "Sex"
 $ : chr "AngPec"
 $ : chr "AMI"
 $ : chr "CAD"
> str(terms(dmod(~.^3, data=cad1, margin=mar))) ## All 3-factor model
List of 4
 $ : chr [1:3] "Sex" "AngPec" "AMI"
 $ : chr [1:3] "Sex" "AngPec" "CAD"
 $ : chr [1:3] "Sex" "AMI" "CAD"
 $ : chr [1:3] "AngPec" "AMI" "CAD"
```

15.2 Altering graphical models

Natural operations on graphical models: add and delete edges

```
> m1 <- dmod(~Sex:Smoker:CAD + CAD:Hyperchol:AMI, data=cad1); m1
Model: A dModel with 5 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :       1293.88 mdim :   13 aic :       1319.88
 ideviance :        112.54 idf  :    8 bic :       1364.91
 deviance  :         16.38 df   :   18
> m2 <- update(m1,items =
               list(dedge=~Hyperchol:CAD, # drop edge
                    aedge=~Smoker:AMI))   # add edge
> par(mfrow=c(1,2)); plot( m1 ); plot( m2 )
```


Sex
Smoker
CAD
Hyperchol
AMI

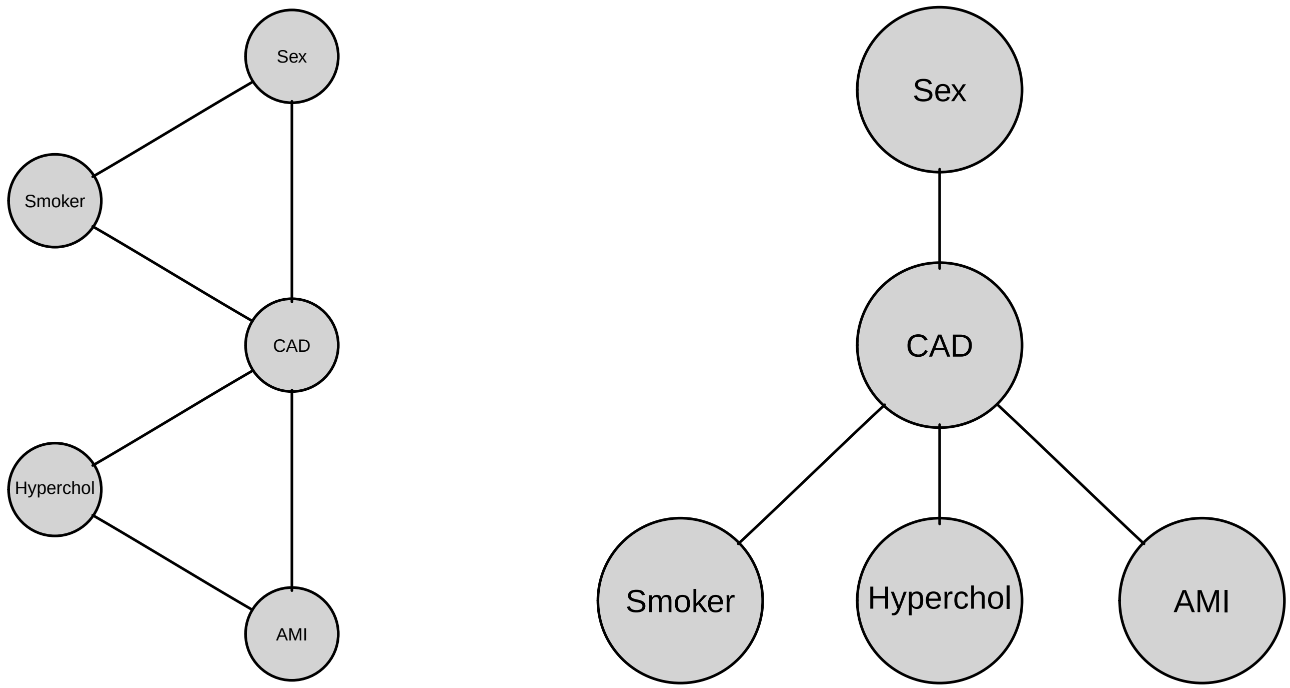

Sex
Smoker
CAD
Hyperchol
AMI

15.3 Model comparison

Models are compared with *compareModels()*.

```
> m1 <- dmod(~Sex:Smoker:CAD + CAD:Hyperchol:AMI, data=cad1); m1
Model: A dModel with 5 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :       1293.88 mdim :   13 aic :      1319.88
 ideviance :        112.54 idf  :    8 bic :      1364.91
 deviance  :         16.38 df   :   18
> m3 <- update(m1, items=list(dedge=~Sex:Smoker+Hyperchol:AMI))
> compareModels( m1, m3 )
Large:
  :"Sex" "Smoker" "CAD"
  :"CAD" "Hyperchol" "AMI"
Small:
  :"Sex" "CAD"
  :"Smoker" "CAD"
  :"CAD" "Hyperchol"
  :"CAD" "AMI"
-2logL:     8.93 df: 4 AIC(k= 2.0):     0.93 p.value: 0.346446
```

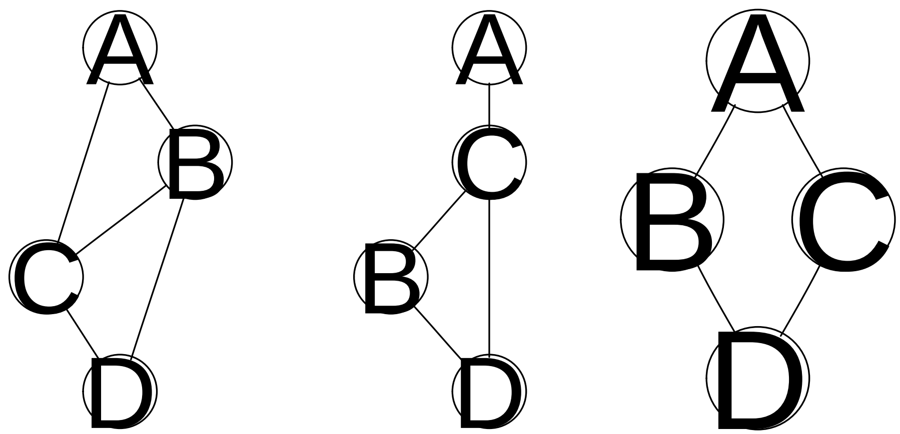
```
> par(mfrow=c(1,2)); plot( m1 ); plot( m3 )
```



15.4 Decomposable models – deleting edges

Result: If $\mathcal{A}_1$ is a decompsable model and we remove an edge $e = \{u, v\}$ which is contained in one clique C only, then the new model $\mathcal{A}_2$ will also be decomposable.

```
> par(mfrow=c(1,3))
> plot(ug(~A:B:C+B:C:D))
> plot(ug(~A:C+B:C+B:C:D))
> plot(ug(~A:B+A:C+B:D+C:D))
```



Left: $\mathcal{A}_1$ – decomposable; Center: dropping $\{A, B\}$ gives decomposable model; Right: dropping $\{B, C\}$ gives non–decomposable model.

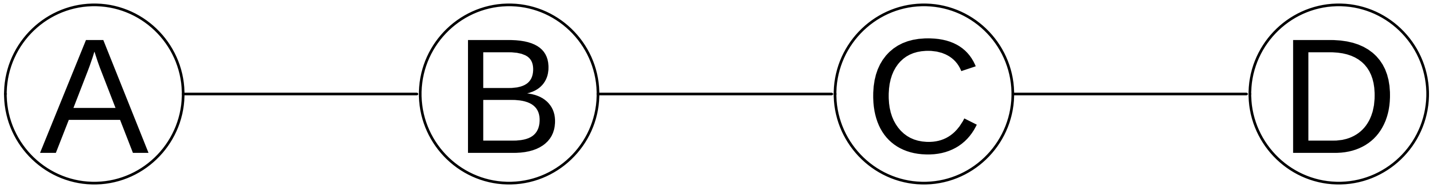
Result: The test for removal of $e = \{u, v\}$ which is contained in one clique C only can be made as a test for $u \perp\!\!\!\perp v | C \setminus \{u, v\}$ in the C−marginal table.

This is done by *ciTest()*. Hence, no model fitting is necessary.

15.5 Decomposable models – adding edges

More tricky when adding edge to a decomposable model

```
> plot(ug(~A:B+B:C+C:D), "circo")
```



Adding $\{A, D\}$ gives non–decomposable model; adding $\{A, C\}$ gives decomposable model.

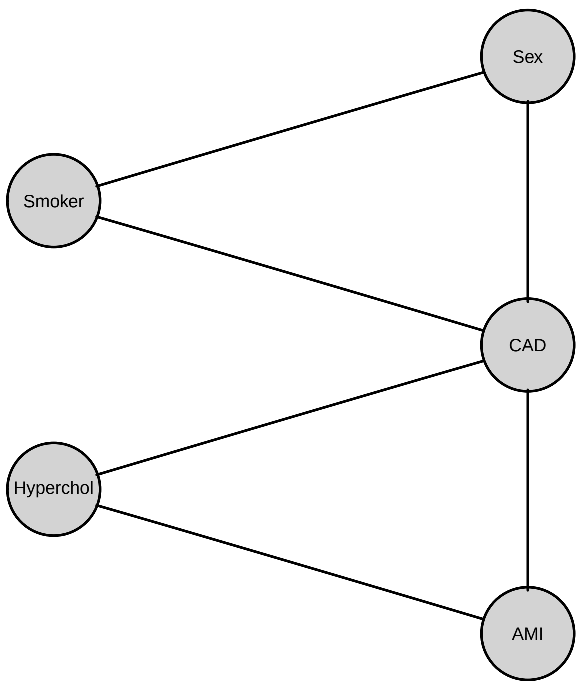
One solution: Try adding edge to graph and test if new graph is decomposable. Can be tested with *maximum cardinality search* as implemented in *mcs()*. Runs in $O(|edges| + |vertices|)$.

```
> UG <- ug(~A:B+B:C+C:D)
> mcs(UG)
[1] "A" "B" "C" "D"
> UG1 <- addEdge("A","D",UG)
> mcs(UG1)
character(0)
> UG2 <- addEdge("A","C",UG)
> mcs(UG2)
[1] "A" "B" "C" "D"
```

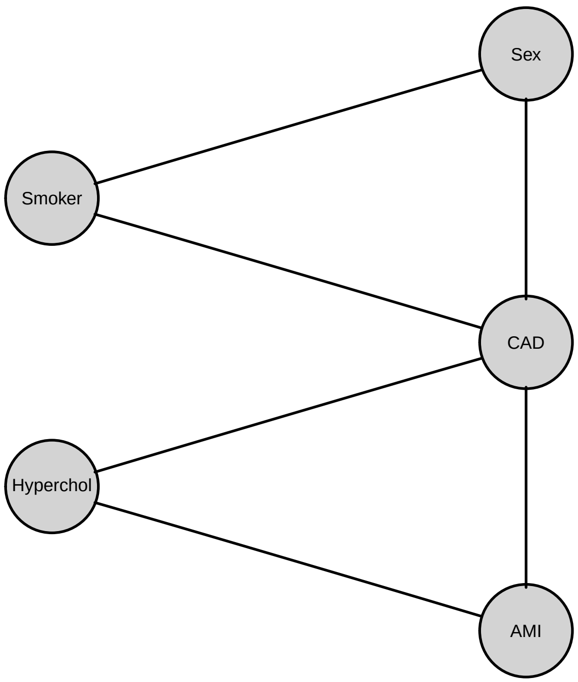
15.6 Test for adding and deleting edges

Done with *testdelete()* and *testadd()*

```
> m1 <- dmod(~Sex:Smoker:CAD + CAD:Hyperchol:AMI, data=cad1)
> plot( m1 )
> testdelete( m1, edge=c("Hyperchol", "AMI") )
dev:    4.981 df:  2 p.value: 0.08288 AIC(k=2.0):    1.0 edge: Hyperch
host:  CAD Hyperchol AMI
Notice: Test performed in saturated marginal model
```



```
> m1 <- dmod(~Sex:Smoker:CAD + CAD:Hyperchol:AMI, data=cad1)
> plot( m1 )
> testadd( m1, edge=c("Smoker", "Hyperchol"))
dev:    1.658 df:  2 p.value: 0.43654 AIC(k=2.0):    2.3 edge: Smoker:
host:  CAD Smoker Hyperchol
Notice: Test performed in saturated marginal model
```



15.7 Model search in log–linear models using **gRim**

Model selection implemented in *stepwise()* function.

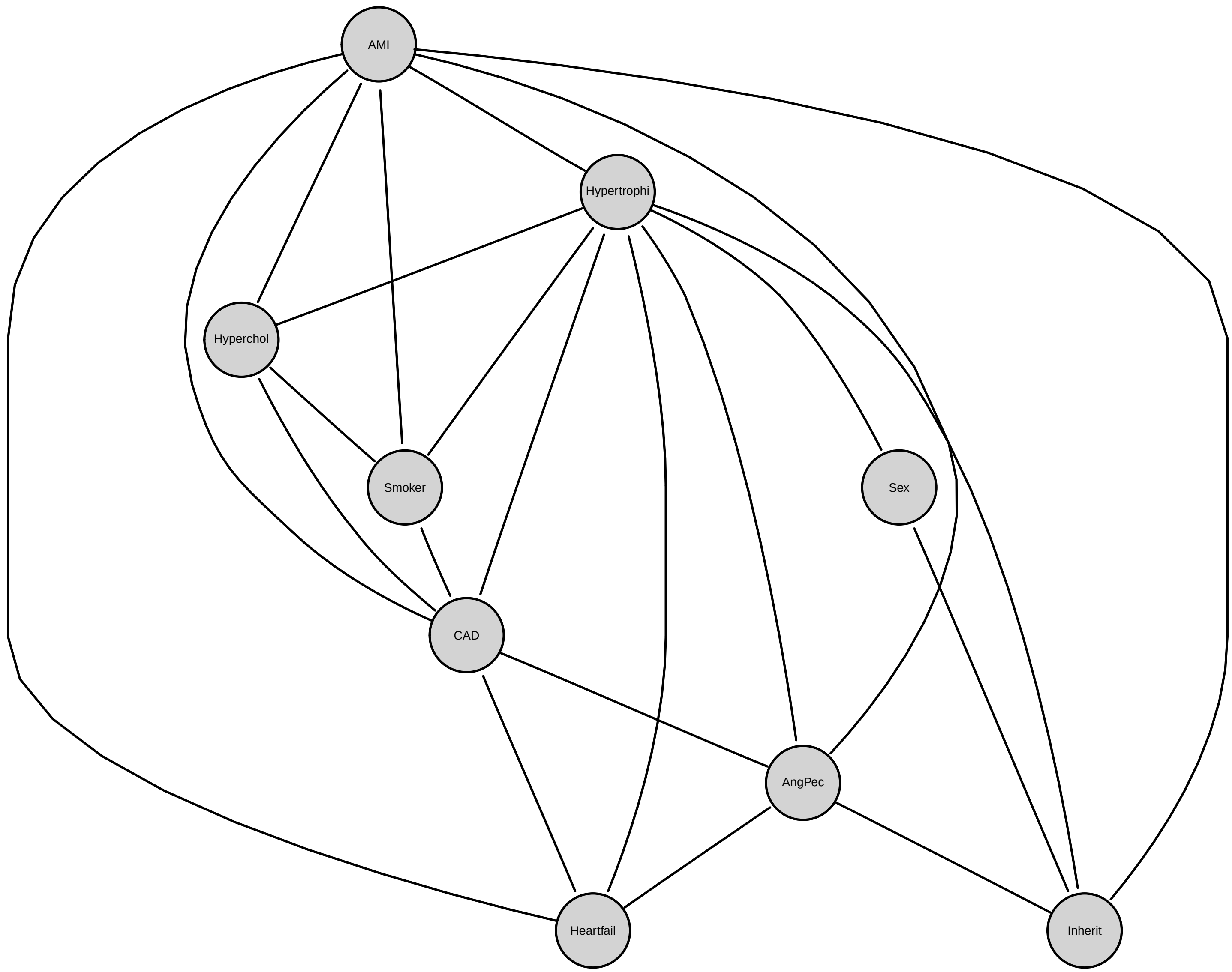
- Backward / forward search (Default: backward)
- Select models based on p–values or AIC(k=2) (Default: AIC(k=2))
- Model types can be "unsrestricted" or "decomposable". (Default is decomposable if initial model is decompsable)
- Search method can be "all" or "headlong". (Default is all)

```
> args(stepwise.iModel)
function (object, criterion = "aic", alpha = NULL, type = "decomposabl
    search = "all", steps = 1000, k = 2, direction = "backward",
    fixinMAT = NULL, fixoutMAT = NULL, details = 0, trace = 2,
    ...)
NULL
```

```
> msat <- dmod( ~.^., data=cad1 )
> mnew1 <- stepwise( msat, 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

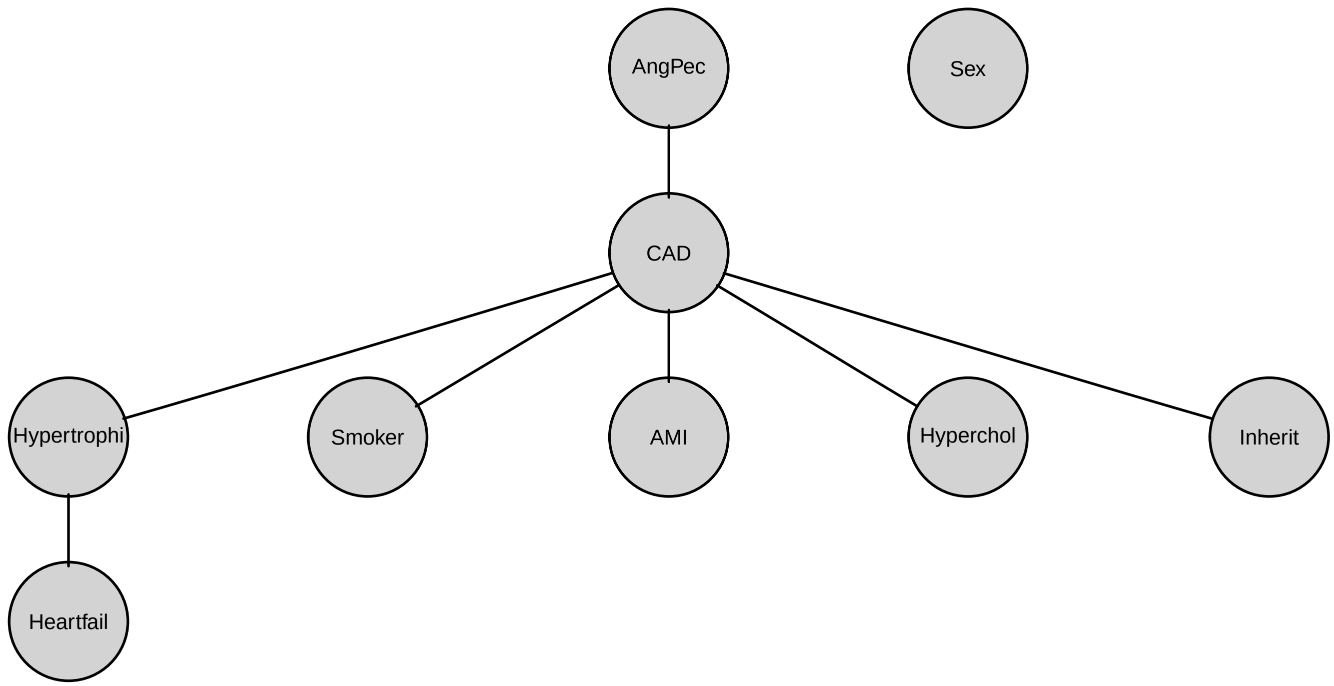
```
> plot( mnew1 )
```


AMI
Hypertrophi
Hyperchol
Smoker
Sex
CAD
AngPec
Heartfail
Inherit

```
> msat <- dmod( ~.^., data=cad1 )
> mnew2 <- stepwise( msat, details=1, k=log(nrow(cad1)) ) # use bic
STEPWISE:
 criterion: aic ( k = 5.46 )
 direction: backward
 type      : decomposable
 search    : all
 steps     : 1000
. BACKWARD: type=decomposable search=all, criterion=aic(5.46), alpha=0
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC -100.0382 Edge deleted: Sex AngPec
  change.AIC -103.1520 Edge deleted: Hyperchol AngPec
  change.AIC  -74.2967 Edge deleted: Smoker AngPec
  change.AIC  -67.8590 Edge deleted: Sex Hyperchol
  change.AIC  -60.3907 Edge deleted: AngPec Hypertrophi
  change.AIC  -51.9489 Edge deleted: Heartfail Hyperchol
  change.AIC  -50.8580 Edge deleted: Sex CAD
  change.AIC  -43.8873 Edge deleted: AngPec Heartfail
  change.AIC  -41.3702 Edge deleted: AMI Sex
  change.AIC  -43.6158 Edge deleted: AMI Heartfail
  change.AIC  -40.2509 Edge deleted: Hyperchol Inherit
  change.AIC  -26.3511 Edge deleted: AngPec AMI
  change.AIC  -31.4947 Edge deleted: Inherit AMI
```

```
change.AIC  -25.5315 Edge deleted: Heartfail CAD
change.AIC  -31.2732 Edge deleted: Inherit Heartfail
change.AIC  -22.9457 Edge deleted: AMI Hypertrophi
change.AIC  -17.9850 Edge deleted: Smoker AMI
change.AIC  -15.7814 Edge deleted: Sex Heartfail
change.AIC  -15.5931 Edge deleted: Smoker Sex
change.AIC  -18.5186 Edge deleted: Inherit Smoker
change.AIC  -13.8092 Edge deleted: Hyperchol Smoker
change.AIC  -12.4648 Edge deleted: AngPec Inherit
change.AIC   -6.5068 Edge deleted: Smoker Heartfail
change.AIC   -9.2031 Edge deleted: Hypertrophi Smoker
change.AIC   -5.9470 Edge deleted: AMI Hyperchol
change.AIC   -5.0227 Edge deleted: Hypertrophi Hyperchol
change.AIC   -4.0234 Edge deleted: Sex Inherit
change.AIC   -6.8882 Edge deleted: Hypertrophi Inherit
change.AIC   -3.1347 Edge deleted: Hypertrophi Sex
```

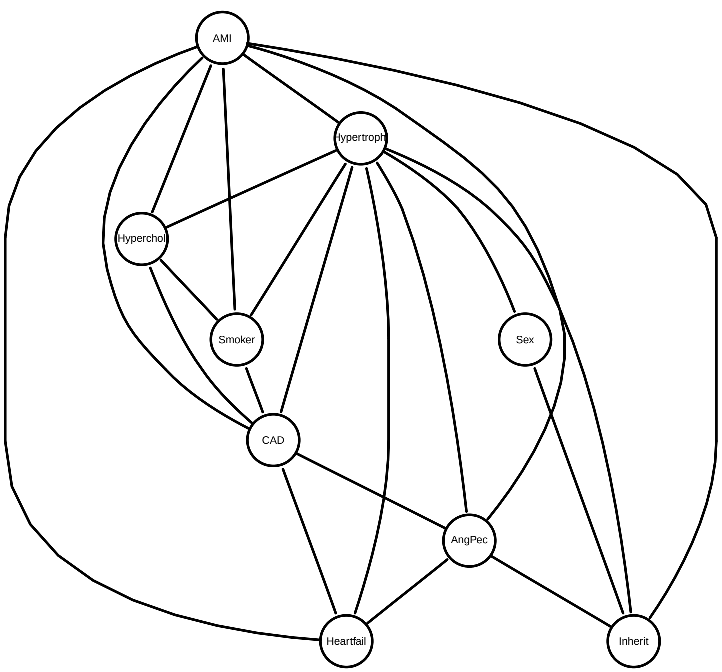
```
> plot( mnew2 )
```

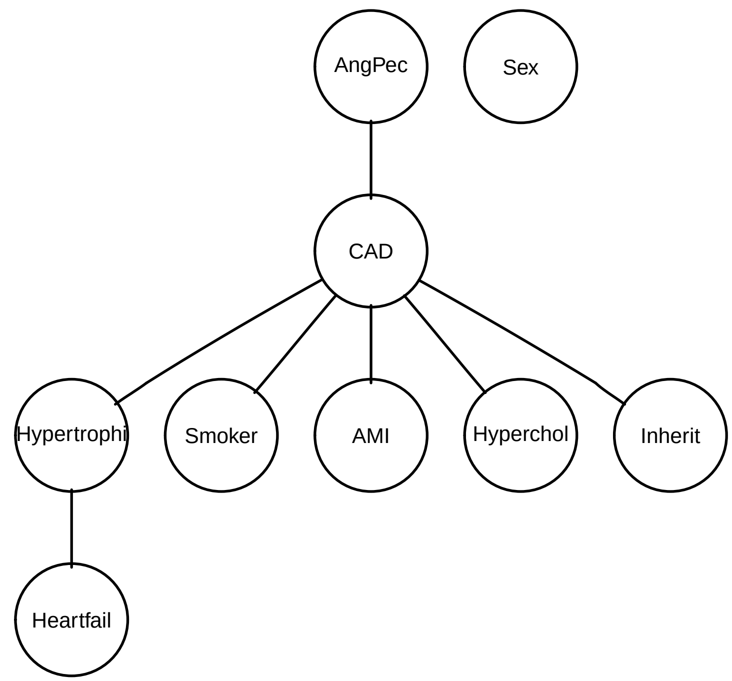

AngPec
Sex
CAD
Hypertrophi
Smoker
AMI
Hyperchol
Inherit
Heartfail

16 From graph and data to network

Create graphs from models:

```
> ug1 <- ugList( terms( mnew1 ) )
> ug2 <- ugList( terms( mnew2 ) )
> par(mfrow=c(1,2)); plot( ug1 ); plot( ug2 )
```





Create Bayesian networks from (graph, data):

```
> bn1 <- compile( grain( ug1, data=cad1, smooth=0.1 )); bn1
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:9] "Hypertrophi" "AMI" "CAD" "Smoker" ...
> bn2 <- compile( grain( ug2, data=cad1, smooth=0.1 )); bn2
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:9] "CAD" "AngPec" "Hypertrophi" "Heartfail" ...
```

```
> querygrain( bn1, "CAD")
$CAD
CAD
   No   Yes
0.546 0.454
> z<-setEvidence( bn1, nodes=c("AngPec", "Hypertrophi"),
                 c("Typical","Yes"))
> # alternative form
> z<-setEvidence( bn1,
                 nslist=list(AngPec="Typical", Hypertrophi="Yes"))
> querygrain( z, "CAD")
$CAD
CAD
   No   Yes
0.599 0.401
```

17 Prediction

Dataset with missing values

```
> data(cad2, package="gRbase")
> dim( cad2 )
[1] 67 14
> head( cad2, 4 )
     Sex   AngPec         AMI QWave QWavecode    STcode STchange
1   Male     None NotCertain    No    Usable    Usable      Yes
2 Female     None NotCertain    No    Usable    Usable      Yes
3 Female     None NotCertain    No Nonusable Nonusable       No
4   Male Atypical   Definite    No    Usable    Usable       No
  SuffHeartF Hypertrophi Hyperchol Smoker Inherit Heartfail CAD
1        Yes          No        No   <NA>      No        No  No
2        Yes          No        No   <NA>      No        No  No
3         No          No       Yes   <NA>      No        No  No
4        Yes          No       Yes   <NA>      No        No  No
```

```
> args(predict.grain)
function (object, response, predictors = setdiff(names(newdata),
    response), newdata, type = "class", ...)
NULL
> 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
```

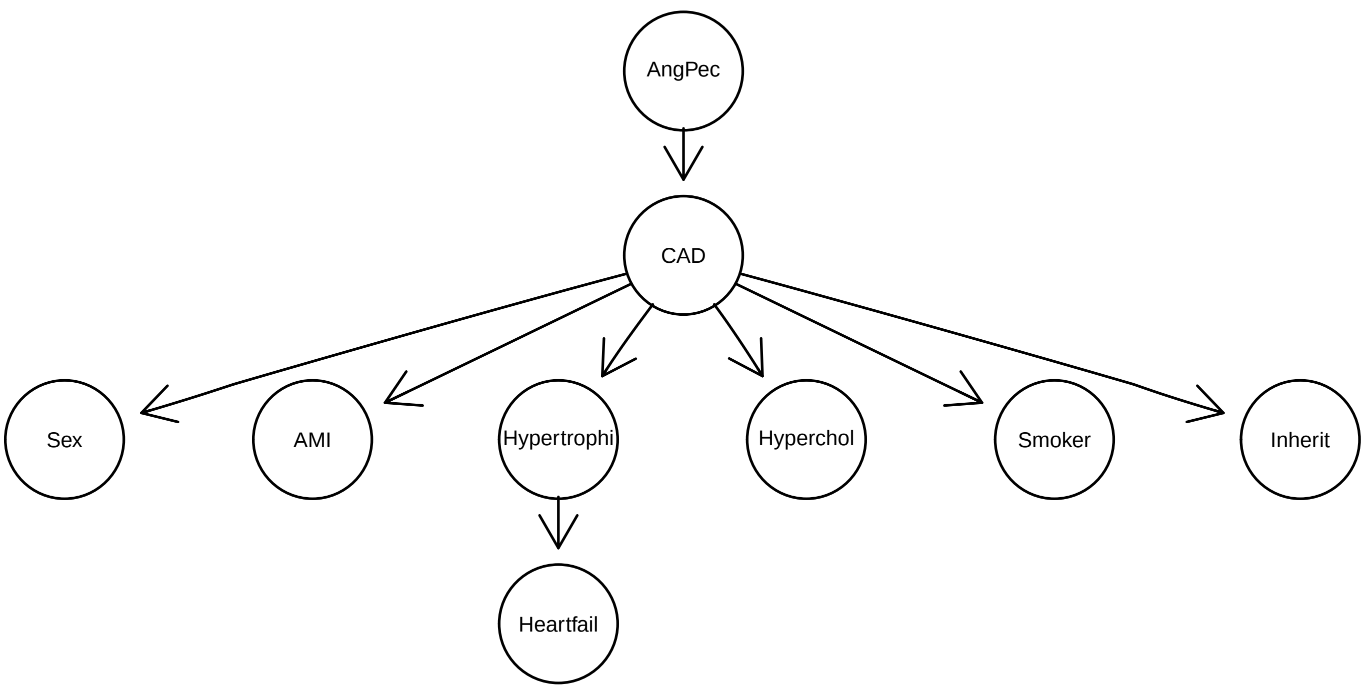
Can be more informative to look at conditional probabilities:

```
> q1 <- predict(bn1, newdata=cad2, response="CAD",
                type="distribution")
> head( q1$pred$CAD )
        No    Yes
[1,] 0.974 0.0258
[2,] 0.974 0.0258
[3,] 0.898 0.1017
[4,] 0.535 0.4651
[5,] 0.787 0.2134
[6,] 0.451 0.5490
> head( p1$pred$CAD )
[1] "No"  "No"  "No"  "No"  "No"  "Yes"
> head( cad2$CAD)
[1] No No No No No No
Levels: No Yes
```

18 Other packages

Model search facilities in **gRim** are limited but the **bnlearn** package contains useful stuff, see `http://www.bnlearn.com/`.

```
> require( bnlearn )
> a = bn.fit(hc( cad1 ), cad1)
> bn = as.grain(a)
> plot(bn)
```



19 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 packages for structural learning, e.g. **bnlearn**.