# Recent developments in R packages for graphical models

**— REVISED February 2013 —**

**Søren Højsgaard**

**Aalborg University, Denmark**

**sorenh@math.aau.dk**

**February 22, 2013**

Computing for Graphical Models

16 December 2011

Royal Statistical Society,

London

Compiled: February 22, 2013 File: GM-RSS-slides.tex

# Contents

## 1 Outline

- Introduce the `gRain` package (GRAphical Independence Networks) for Bayesian networks
- Conditional independence restrictions, dependency graphs, message passing
- Log–linear, graphical and decomposable models for contingency tables
- Introduce the `gRim` package (GRaphical Independence Models)
- Convert decomposable model to Bayesian network.
- The `gRbase` package and some graph algorithms

## 2 Graphs for graphical models

Three different representations of graphs:

```
form <- ~a:b+b:c
ug.NEL  <- ug(form, result="NEL")    # graphNEL (DEFAULT)
ug.MAT  <- ug(form, result="matrix")
ug.SMAT <- ug(form, result="Matrix") # Sparse dgCMatrix
ug.IG   <- ug(form, result="igraph")
```

- The packages `graph` and `RBGL` based on `graphNEL` (Node-EdgeList representation)
- Package `igraph` has a similar internal representation.

```
ug.NEL
```

```
A graphNEL graph with undirected edges
Number of Nodes = 3
Number of Edges = 2
```

```
nodes(ug.NEL)
```

```
[1] "a" "b" "c"
```

```
str(edges(ug.NEL))
```

```
List of 3
 $ a: chr "b"
 $ b: chr [1:2] "c" "a"
 $ c: chr "b"
```

```
ug.MAT
```

```
  a b c
a 0 1 0
b 1 0 1
c 0 1 0
```

```
ug.SMAT
```

```
3 x 3 sparse Matrix of class "dgCMatrix"
  a b c
a . 1 .
b 1 . 1
c . 1 .
```

```
ug.IG
```

```
IGRAPH UNW- 3 2 --
+ attr: name (v/c), label (v/c), weight (e/n)
```

```
V(ug.IG)
```

```
Vertex sequence:
[1] "a" "b" "c"
```

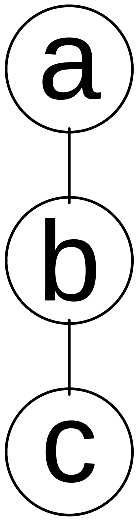
```
E(ug.IG)
```

```
Edge sequence:

[1] b -- a
[2] c -- b
```

- There are `plot` methods for `graphNEL`s and for `igraph`s.
- Graph rendering leaves something to be desired...

```
plot(ug.NEL)
```



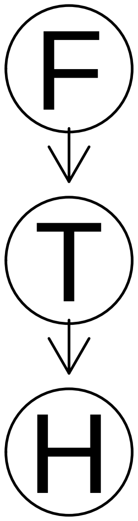
## 3 Bayesian networks – the `gRain` package

Consider the following narrative:

> Having flu (F) may cause an elevated body temperature (T) (fever). An elevated body temperature may cause a headache (H).

Illustrate this narrative by directed acyclic graph (or DAG ):

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



- We have a universe  consisting of the variables $V = \{F, T, H\}$ which all have a finite state space . (Here all are binary).
- Corresponding to $V$ there is a random vector $X = X_V = (X_F, X_T, X_H)$ where $x = x_V = (x_F, x_T, x_H)$ denotes a specific configuration .
- For $A \subset V$ we have the random vector $X_A = (X_v; v \in A)$ where a configuration is denoted $x_A$.
- We define a joint pmf for $X$ as

$$p_X(x) = p_{X_F}(x_F) p_{X_T|X_F}(x_T|x_F) p_{X_H|X_T}(x_H|x_T) \tag{1}$$

- We allow a simpler notation: Let $A$ and $B$ be disjoint subsets of $V$. We may then use one of the forms:

$$p_{X_A|X_B}(x_A|x_B) = p_{A|B}(x_A|x_B) = p(x_A|x_B) = p(A|B)$$

Hence (1) may be written

$$p(V) = p(F)p(T|F)p(H|T)$$

- Notice: By definition of conditional probability we have from Bayes formula that

$$p(V) \equiv p(F,T,H) = p(F)p(T|F)p(H|T,F)$$

- So the fact that in

$$p(V) = p(F)p(T|F)p(H|T)$$

  we have $p(H|T)$ rather than $p(H|T,F)$ reflects the model assumption that if temperature (fever status) is known then knowledge about flu will provide no additional information about headache.

- We say that headache is conditionally independent of flu given temperature.
- Given a finding or evidence that a person has headache we may now calculate e.g. the probability of having flu, i.e. $p(F = yes|H = yes)$.
- In this small example we can compute everything in a brute force way using table operation functions from `gRbase`.

## 3.1 Specification of conditional probability tables

We may specify $p(F)$, $p(T|F)$ and $p(H|T)$ as tables with **parray()** (using **array()** is an alternative), where "yes" is coded as $1$ and "no" as $2$.

```
p.F   <- parray("F", levels=2, values=c(.01,.99))
```

```
F
  F1   F2
0.01 0.99
```

```
p.TgF <- parray(c("T","F"), levels=c(2,2), values=c(.95,.05, .01,.99))
```

```
    F
T      F1   F2
  T1 0.95 0.01
  T2 0.05 0.99
```

```
p.HgT <- parray(c("H","T"), levels=c(2,2), values=c(.8,.2, .1,.9))
```

```
    T
H     T1  T2
  H1 0.8 0.1
```

```
H2 0.2 0.9
```

## 3.2 Brute force computations of conditional probabilities

Functions **tableMult()** , **tableDiv()** and **tableMargin()** are useful.

1) First calculate joint distribution:

```
p.V <- tableMult(tableMult(p.F, p.TgF), p.HgT)
head(as.data.frame.table(p.V))
```

```
   H  T  F    Freq
1 H1 T1 F1 0.00760
2 H2 T1 F1 0.00190
3 H1 T2 F1 0.00005
4 H2 T2 F1 0.00045
5 H1 T1 F2 0.00792
6 H2 T1 F2 0.00198
```

2) Then calculate the marginal distribution

```
p.FT <- tableMargin(p.V, margin=c('F','T'))
as.data.frame.table(p.FT)
```

```
   F  T   Freq
1 F1 T1 0.0095
2 F2 T1 0.0099
```

```
3 F1 T2 0.0005
4 F2 T2 0.9801
```

3) Then calculate conditional distribution

```
p.T <- tableMargin(p.FT, margin='T')
p.FgT <- tableDiv(p.FT, p.T)
p.FgT
```

```
    F
T               F1        F2
  T1 0.4896907216 0.5103093
  T2 0.0005098919 0.9994901
```

So $p(F = yes|H = yes) = 0.500$.

However, this scheme is computationally prohibitive in large networks: With $80$ variables each with $10$ the total state space is $10^{80}$ – the estimated number of atoms in the universe...

## 3.3 Bayesian Network (BN) basics

- A Bayesian network is a often understood to be graphical model based on a directed acyclic graph (a DAG ).
- A BN typically will typically satisfy conditional independence restrictions which enables computations of updated probabilities for states of unobserved variables to be made very efficiently .
- The DAG only is used to give a simple and transparent way of specifying a probability model.
- The computations are based on exploiting conditional independencies in an undirected graph.
- Therefore, methods for building undirected graphical models can just as easily be used for building BNs.

## 3.4 The chest clinic narrative

Lauritzen and Spiegehalter (1988) presents the following narrative:

> “Shortness–of–breath (dyspnoea) may be due to tuberculosis, lung cancer or bronchitis, or none of them, or more than one of them.
>
> A recent visit to Asia increases the chances of tuberculosis, while smoking is known to be a risk factor for both lung cancer and bronchitis.
>
> The results of a single chest X–ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of dyspnoea.”

The universe is

$$V = \{\texttt{Asia}, \texttt{Tub}, \texttt{Smoke}, \texttt{Lung}, \texttt{Either}, \texttt{Bronc}, \texttt{X-ray}, \texttt{Dysp}\}$$

A formalization of this narrative is as follows: The DAG in Figure 1 corresponds to a factorization of the joint probability function as

$$p(V) = p(A)p(T|A)p(S)p(L|S)p(B|S)p(E|T,L)p(D|E,B)p(X|E). \qquad (2)$$

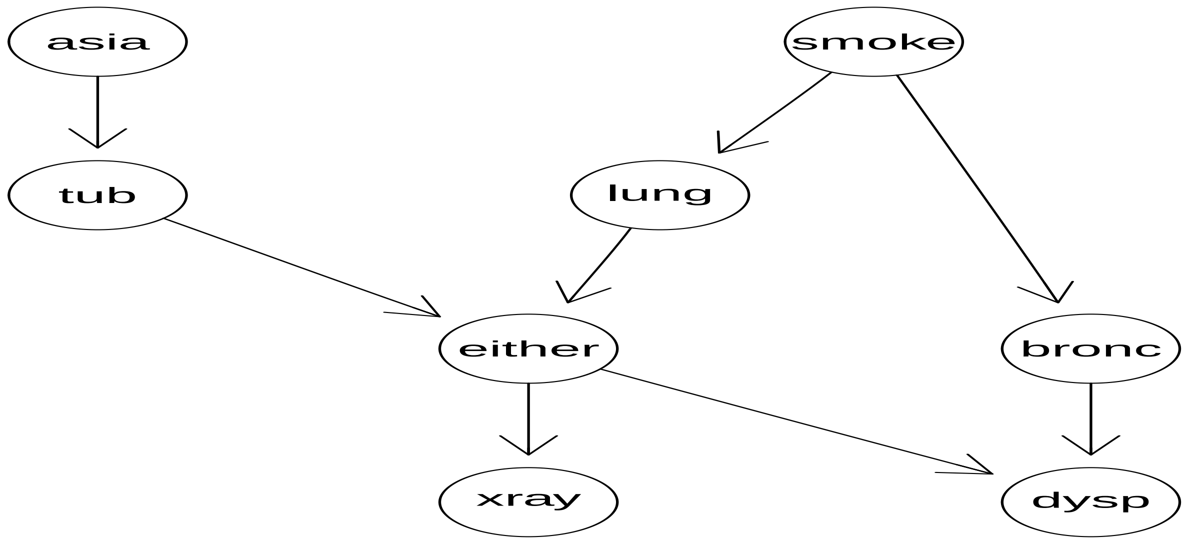


Figure 1: The directed acyclic graph corresponding to the chest clinic example.

## 3.5 Findings and queries

- Suppose we are given the finding that a person has recently visited Asia and suffers from dyspnoea, i.e. $A = \text{yes}$ and $D = \text{yes}$. Generally denote findings as $E = e^*$
- Interest may be in the conditional distributions $p(L|e^*)$, $p(T|e^*)$ and $p(B|e^*)$, or possibly in the joint (conditional) distribution $p(L, T, B|e^*)$.
- Interest might also be in calculating the probability of a specific event, e.g. the probability of seeing a specific evidence, i.e. $p(E = e^*)$.
- `gRain` does this by using the Lauritzen–Spiegelhalter (1988) algorithm.

## 3.6 The chest clinic

Specify chest clinic network.

```
yn <- c("yes","no")
a    <- cptable(~asia, values=c(1,99),levels=yn)
t.a  <- cptable(~tub+asia, values=c(5,95,1,99),levels=yn)
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)
## Some initial compilation steps:
plist <- compileCPT(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
## Build network
bnet    <- grain(plist)
bnet
```

```
Independence network: Compiled: FALSE Propagated: FALSE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" ...
```

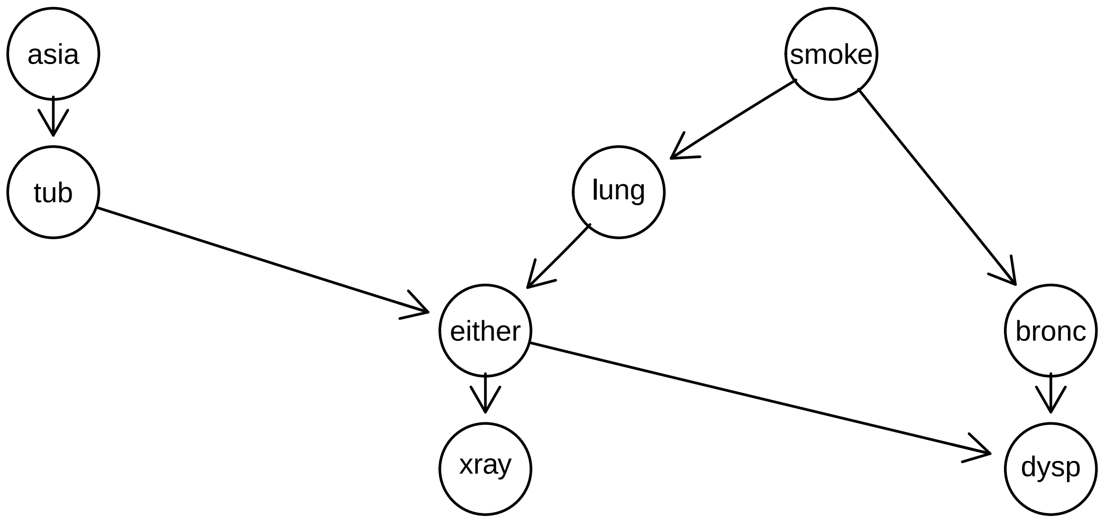
```
plist
```

```
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 )
```

```
plist$tub
```

```
NULL
```

```
plot(bnet)
```



## 3.7 Queries – getting beliefs

```
querygrain(bnet, nodes=c('lung', 'tub', '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
```

## 3.8 Setting findings – and getting updated beliefs

```
(bnet.f <- setFinding(bnet, nodes=c('asia', 'dysp'), state=c('yes','yes')))
```

```
Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" ...
  Findings: chr [1:2] "asia" "dysp"
```

```
querygrain(bnet.f, nodes=c('lung', 'tub', 'bronc'))
```

```
$tub
tub
       yes         no
0.08775096 0.91224904

$lung
lung
       yes         no
0.09952515 0.90047485

$bronc
bronc
      yes        no
```

```
0.8114021 0.1885979
```

```
querygrain(bnet.f, nodes=c('lung', 'tub', 'bronc'), type='joint')
```

```
, , bronc = yes

     tub
lung          yes          no
  yes 0.003149038 0.05983172
  no  0.041837216 0.70658410

, , bronc = no

     tub
lung          yes          no
  yes 0.001827219 0.03471717
  no  0.040937491 0.11111605
```

## 3.9 Probability of a finding

```
getFinding(bnet.f)
```

```
Finding:
     variable state
[1,] asia     yes
[2,] dysp     yes
Pr(Finding)= 0.004501375
```

```
pFinding(bnet.f)
```

```
[1] 0.004501375
```

## 4 Under the hood of gRain – and on the way to gRim

Efficient compuations in BNs are based on exploiting conditional independence restrictions.

- $X$ and $Y$ are conditionally independent given $Z$ (written $X \perp\!\!\!\perp Y|Z$) if

$$p(x,y|z) = p(x|z)p(y|z)$$

  – or equivalently

$$p(y|x,z) = p(y|z)$$

- So if $Z=z$ is known then knowledge of $X$ will provide no additional knowledge of $Y$.

- A general condition is the factorization criterion : $X \perp\!\!\!\perp Y|Z$ if

$$p(x,y,z) = g(x,z)h(y,z)$$

  for non–negative functions $g()$ and $h()$.

## 4.1 Dependence graph

Given variables $V$, let $\mathcal{A} = \{a_1, \dots, a_Q\}$ be a collection of subset of $V$.

Suppose

$$p(x) = \prod_{a \in \mathcal{A}} \phi_a(x_a)$$

where $\phi_a()$ is a non–negative function of $x_a$.

The dependence graph for $p$ has vertices $V$ and undirected edges given as follows:

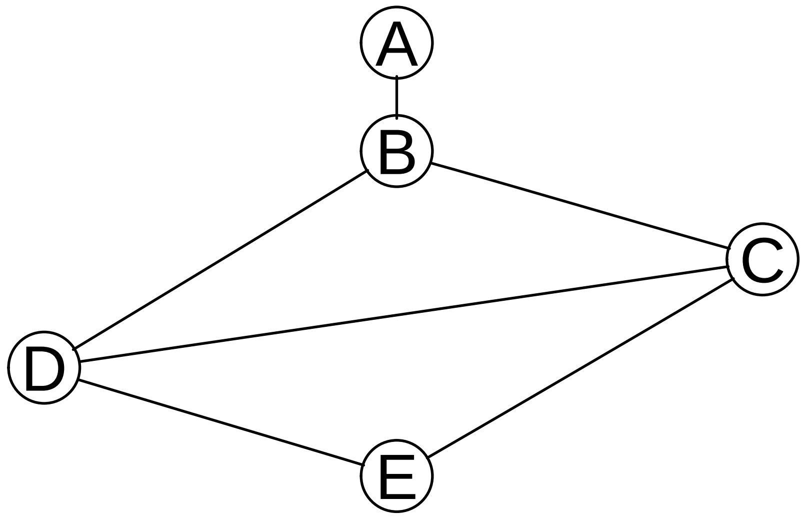
There is an edge between $\alpha$ and $\beta$ iff $\{\alpha, \beta\}$ is in one of the sets $a \in \mathcal{A}$.

Suppose

$$p(x) = \phi_{AB}(x_{AB})\psi_{BCD}(x_{BCD})\psi_{CDE}(x_{CDE})$$

Then the dependence graph for $p$ is given as follows:

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



## 4.2 Reading conditional independencies – global Markov property

Conditional independencies can be read off the dependence graph:

- Recall basic factorization $p$:

$$p(x) = \phi_{AB}(x_{AB})\psi_{BCD}(x_{BCD})\psi_{CDE}(x_{CDE})$$

- Recall factorization criterion:

$$X \perp\!\!\!\perp Y|Z \text{ if } p(x,y,z) = g(x,z)h(y,z)$$

- Global Markov Property : If $X$ and $Y$ are separated by $Z$ in the dependence graph $\mathcal{G}$ then $X \perp\!\!\!\perp Y|Z$.
- Example: $(D,E) \perp\!\!\!\perp A|(B,C)$:

  Proof:

$$p(x) = \left[\phi_{AB}(x_{AB})\psi_{BCD}(x_{BCD})\right]\left[\psi_{CDE}(x_{CDE})\right] = g(x_{ABCD})h(x_{CDE})$$

## 4.3 Dependence graph for chest clinic example

- Recall chest clinic DAG–factorization

$$p(V) = p(A)p(T|A)p(S)p(L|S)p(B|S)p(E|T,L)p(D|E,B)p(X|E).$$

- Think of conditional probilities as potentials and rewrite as:

$$p(V) = \psi(A)\psi(T,A)\psi(S)\psi(L,S)\psi(B,S)\psi(E,T,L)\psi(D,E,B)\psi(X,E).$$

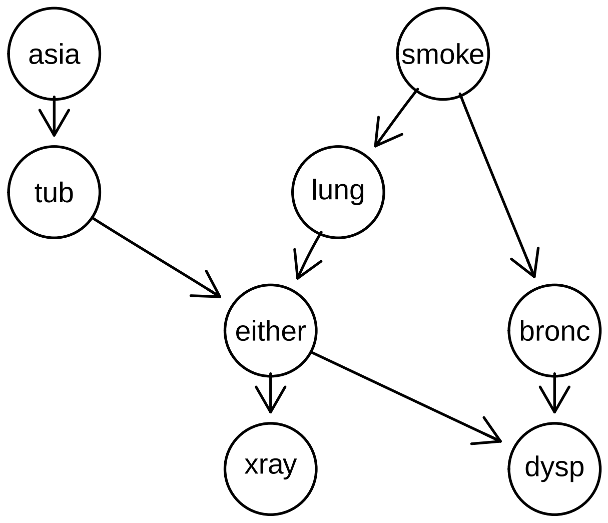
- Next, absorb lower order terms into higher order terms:

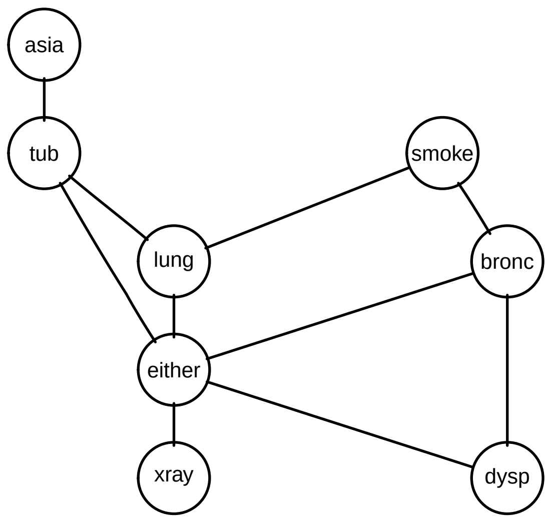
$$p(V) = \psi(T,A)\psi(L,S)\psi(B,S)\psi(E,T,L)\psi(D,E,B)\psi(X,E).$$

- The dependence graph of the last form is the moral graph of the DAG.

Given DAG, the moral graph is obtained by 1) marrying parents and 2) dropping directions – called moralization ; **moralize()** does this.

```
par(mfrow=c(1,2))
plot(bnet$dag)
plot(moralize(bnet$dag))
```



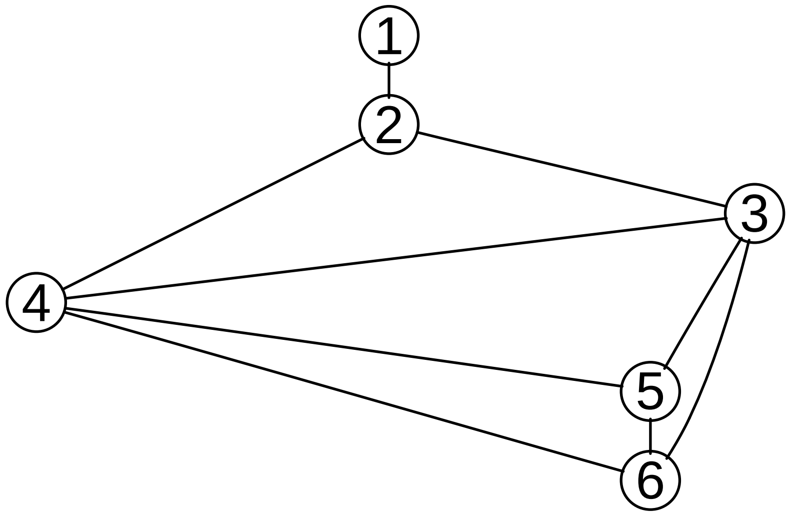


## 4.4 Graphs and cliques

A clique of a graph is a maximal complete subgraph .

```
plot((g<-ug(~1:2+2:3:4+3:4:5:6)))
str(maxClique(g)$maxCliques)
```

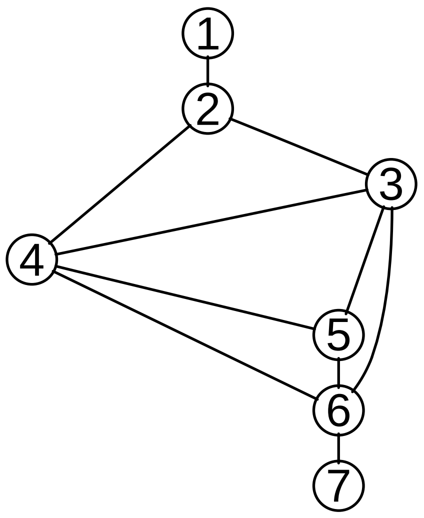
```
List of 3
 $ : chr [1:4] "3" "4" "5" "6"
 $ : chr [1:3] "3" "4" "2"
 $ : chr [1:2] "1" "2"
```

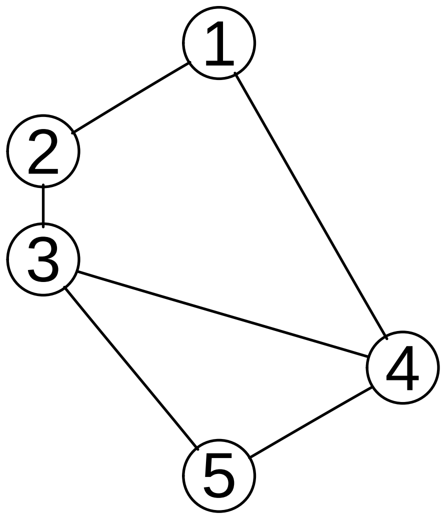


## 4.5 Decomposable graphs

A graph is decomposable (or triangulated ) iff it contains no cycles of length $\geq 4$.

```
par(mfrow=c(1,2))
plot((g1<- ug(~1:2+2:3:4+3:4:5:6+6:7)))
plot((g2<- ug(~1:2+2:3+3:4:5+4:1)))
```





## Decomposability can be checked with Maximum Cardinality Search (**mcs()** ):

```
mcs(g1)
```

```
[1] "1" "2" "3" "4" "5" "6" "7"
```

```
mcs(g2)
```

```
character(0)
```

## 4.6 The key computations with BNs: message passing

- Suppose

$$p(x) = \prod_{C:cliques} \psi_C(x_C)$$

where $C$ are the cliques of a decomposable graph .

- We may write $p$ in a clique potential representation

$$p(x) = \frac{\prod_{C:cliques} \psi_C(x_C)}{\prod_{S:separators} \psi_S(x_S)}$$

- The terms are called potentials ; the representation is not unique.
- Potential representation easy to obtain:
  - Set all $\psi_C(x_C) = 1$ and all $\psi_S(x_S) = 1$
  - Assign each conditional $p(x_v|x_{pa(v)})$ to a potential $\psi_C$ for a clique $C$ containing $v \cup pa(v)$ by

$$\psi_C(x_C) \leftarrow \psi_C(x_C)p(x_v|x_{pa(v)})$$

- Using local computations (multiplying and dividing low–dimensional tables) we can manipulate the potentials to obtain clique marginal representation :

$$p(x) = \frac{\prod_{C:cliques} p_C(x_C)}{\prod_{S:separators} p_S(x_S)}$$

  1. First until the potentials contain the clique and separator marginals, i.e. $\psi_C(x_C) = p_C(x_C)$.
  2. Next until the potentials contain the clique and separator marginals conditional on a certain set of findings, i.e. $\psi_C(x_C, e^*) = p_C(x_C|e^*)$.

- Done by message passing : Sending messages between neighbouring cliques in decomposable graph – in a particular order – a rip ordering.

- Notice: We do not want to carry out the multiplication above. Better to think about that we have a representation of $p$ as

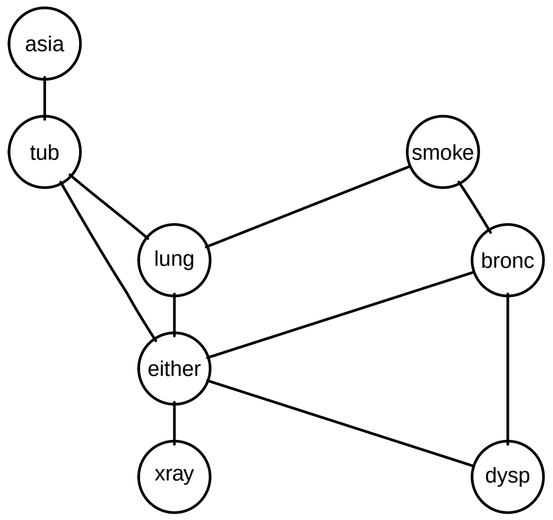
$$p \equiv \{p_C, p_S; C : cliques, S : separators\}$$

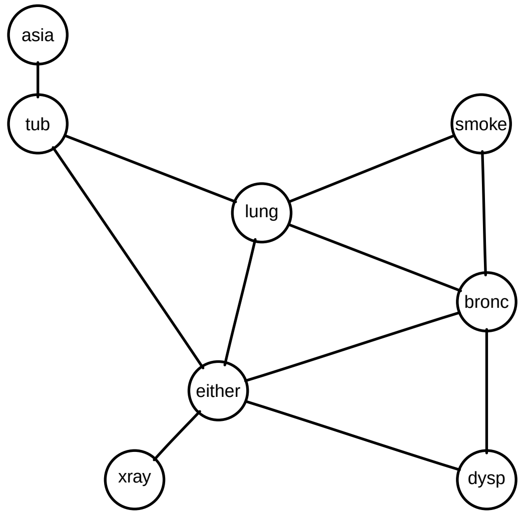
## 4.7 Triangulation

The dependence graph for the chest clinic example is not decomposable (it contains 4–cycles) so the message passing scheme is not directly applicable.

But, we can add edges, so called fill–ins to the the dependence graph to make the graph decomposable. This is called triangulation :

```
par(mfrow=c(1,2))
plot((mdag <- moralize(bnet$dag)))
plot((tmdag <- triangulate(moralize(bnet$dag))))
```





DAG:

$$p(V) = p(A)p(T|A)p(S)p(L|S)p(B|S)p(D|E,B)p(E|T,L)p(X|E).$$

Dependence graph (moral graph):

$$p(V) = \psi(T,A)\psi(L,S)\psi(B,S)\psi(D,E,B)\psi(E,T,L)\psi(X,E).$$

Triangulated graph:

$$p(V) = \psi(T,A)\psi(L,S,B)\psi(L,E,B)\psi(D,E,B)\psi(E,T,L)\psi(X,E)$$

where

$$\psi(L,S,B) = \psi(L,S)\psi(B,S) \quad \phi(L,E,B) \equiv 1$$

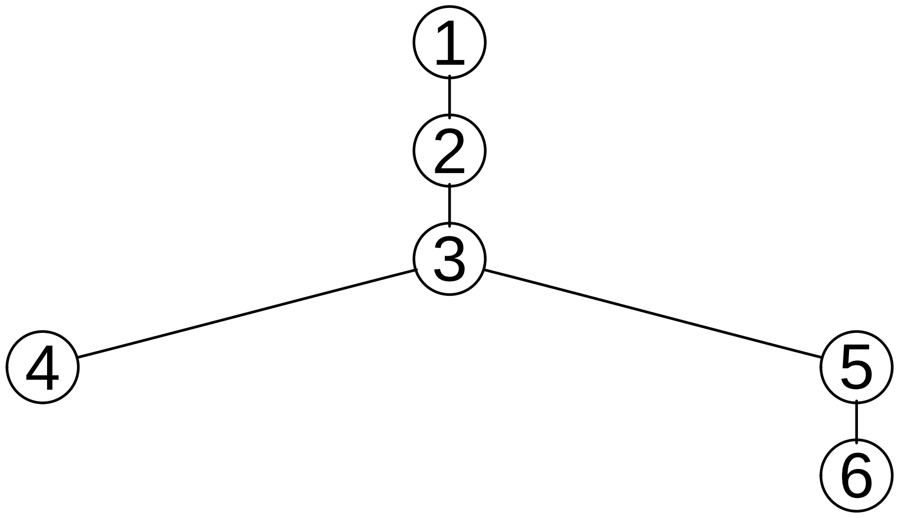
Notice: We have not changed the fundamental model by these steps, but some conditional independencies are concealed in the triangulated graph.

But the triangulated graph factorization allows efficient calculations. $\checkmark$

```
(chest.rip <- rip(tmdag))
```

```
cliques
  1 : tub asia
  2 : either tub lung
  3 : bronc lung either
  4 : smoke lung bronc
  5 : dysp bronc either
  6 : xray either
separators
  1 :
  2 : tub
  3 : lung either
  4 : lung bronc
  5 : bronc either
  6 : either
parents
  1 : 0
  2 : 1
  3 : 2
  4 : 3
  5 : 3
  6 : 5
```

```
plot(chest.rip)
```



## 4.8 Fundamental operations in gRain

Fundamental operations in gRain so far:

- Network specification: **grain()** Create a network from list of conditional probability tables; and do a few other things.
- Set findings: **setFinding()** : Set the values of some variables.
- Ask queries: **querygrain()** : Get updated beliefs (conditional probabilities given findings) of some variables

Under the hood there are two additional operations:

- Compilation: **compile()** Create a clique potential representation (and a few other steps)
- Propagation: **propagate()** Turn clique potentials into clique marginals.

These operations must be made before **querygrain()** can be called but **querygrain()** will make these operations if necessary.

## 4.9 Summary of the BN part

- We have used a DAG for specifying a complex stochastic model through simple conditional probabilities

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

- Afterwards we transfer the model to a factorization over the cliques of a decomposable graph

$$p(V) = \{\prod_{C:cliques} \psi_C(C)\}/\{\prod_{S:separators} \psi_S(S)\}$$

- We then forget about the DAG part and the conditional probability tables.
- It is through message passing between cliques of the decomposable graph that the efficient computation of probabilities takes place.
- Therefore, we may skip the DAG part and find the decomposable graph and corresponding clique potentials from data.

## 5 Contingency tables

In a study of lizard behaviour, characteristics of 409 lizards were recorded, namely species (S), perch diameter (D) and perch height (H).

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

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

```
dim(lizardRAW)
```

```
[1] 409   3
```

We summarize data in a contingency table with cells $(dhs)$ and counts $n_{dhs}$ given by:

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

```
, , species = anoli

     height
diam  >4.75 <=4.75
  <=4    32     86
  >4     11     35

, , species = dist

     height
diam  >4.75 <=4.75
  <=4    61     73
  >4     41     70
```

## 5.1 Log–linear models

- We model the cell probabilities $p_{dhs}$.
- Commonly done by a hierarchical expansion of log–cell–probabilities 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 interaction terms to zero.
- Must follow the principle that 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 $\gamma_{dhs}^{DHS} = 0$.
- 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

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

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

$$p_{dhs} = \psi^{DS}_{ds} \psi^{HS}_{hs}$$

- The factorization criterion  gives directly that $D \perp\!\!\!\perp H | S$.

- More generally the generating class of a log–linear model is a set $\mathcal{A} = \{a_1, \dots, a_Q\}$ where $a_q \subset V$.
- This corresponds to

$$p(x) = \prod_{a \in \mathcal{A}} \phi_a(x_a)$$

  where $\phi_a$ is a potential, a function that depends on $x$ only through $x_a$.
- Under multinomial sampling the likelihood is

$$L = \prod_{x} p(x)^{n(x)} = \prod_{a \in \mathcal{A}} \prod_{x_A} \phi_a(x_a)^{n(x_a)}$$

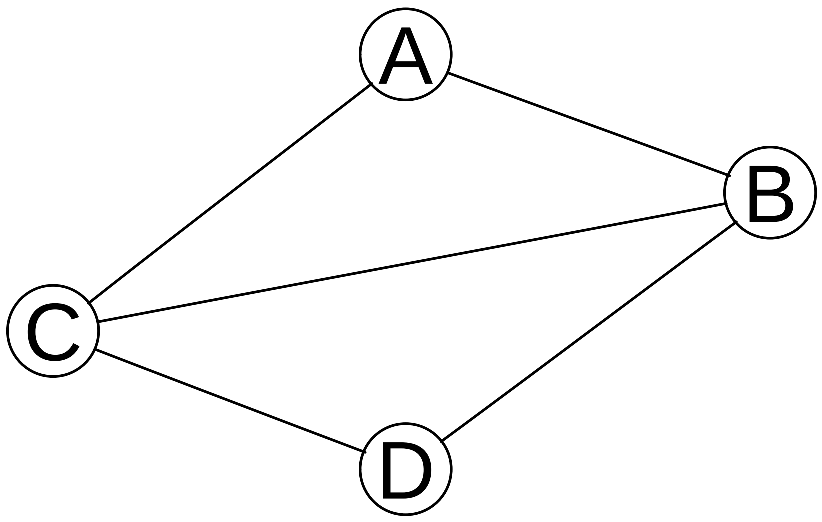
- In `gRim`, the MLE $\hat{p}$ is found by IPS (iterative proportional scaling) as implemented in **loglin()** .

## 5.2 Graphical models

A hierarchical log–linear model with generating class $\mathcal{A}$ is graphical if $\mathcal{A}$ are the cliques of the dependence graph.

EXAMPLE: $\mathcal{A}_1 = \{ABC, BCD\}$ is graphical but $\mathcal{A}_2 = \{AB, AC, BCD\}$ is not graphical. Both have dependence graph with cliques $\mathcal{A}_1$.

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

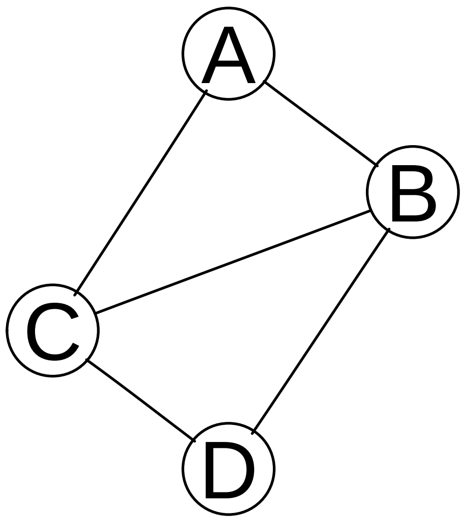


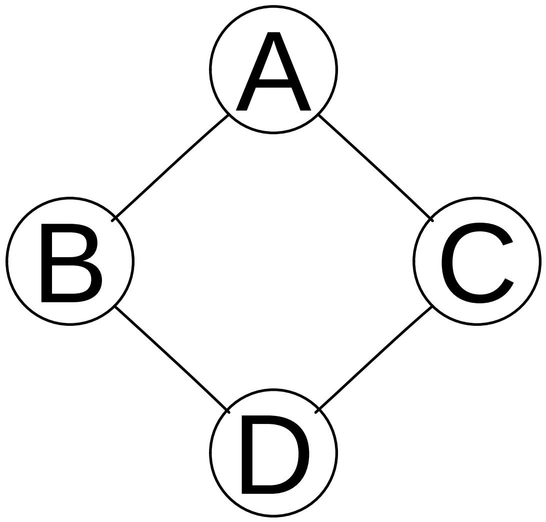
## 5.3 Decomposable models

A graphical log–linear model is decomposable if the models dependence graph is decomposable.

EXAMPLE: $\mathcal{A}_1 = \{ABC, BCD\}$ is decomposable; $\mathcal{A}_2 = \{AB, AC, BD, CD\}$ is not.

```
par(mfrow=c(1,2))
plot((g1<-ug(~A:B:C+B:C:D)))
plot((g2<-ug(~A:B+A:C+B:D+C:D)))
```





## 5.4 ML estimation in decomposable models

- For a decomposable model, the MLE can be found in closed form as

$$\hat{p}(x) = \frac{\prod_{C:cliques} \hat{p}_C(x_C)}{\prod_{S:separators} \hat{p}_S(x_S)} \tag{3}$$

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

- Consider the lizard data and the model $\mathcal{A} = \{[DS][HS]\}$. The MLE is

$$\hat{p}_{dhs} = \frac{(n_{ds}/n)(n_{hs}/n)}{n_s/n} = \frac{n_{ds}n_{hs}}{nn_s}$$

- The result (3) is IMPORTANT in connection with Bayesian networks, because we obtain a clique potential representation of $p$ directly.

- Hence if we find a decomposable graphical model then we can convert this to a Bayesian network.

## 6 Testing for conditional independence

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

## The `set` argument can be given in different forms:

Alternative forms are available:

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

## 6.1 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 be formulated as a test of the conditional independence $u \perp\!\!\!\perp v | S$ in a three way table.

The deviance decomposes into independent contributions from each stratum:

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

where the contribution $D_s$ from the $s$th 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))
```

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

```
cit$slice
```

```
  statistic   p.value df species
1 0.1779795 0.6731154  1   anoli
2 1.8476671 0.1740550  1    dist
```

The $s$th 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.

## 6.2 Monte Carlo tests*

An alternative to the asymptotic $\chi^2$ test is to determine the reference distribution using Monte Carlo methods.

The marginal totals are sufficient statistics under the null hypothesis, and in a conditional test the test statistic is evaluated in the conditional distribution given the sufficient statistics.

Hence one can generate all possible tables with those given margins, calculate the desired test statistic for each of these tables and then see how extreme the observed test statistic is.

A Monte Carlo approximation is to randomly generate large number of tables with the given margins and then proceed as above.

This is called a Monte Carlo exact test and it provides a Monte Carlo $p$–value .

```
ciTest(lizard, set=~diam+height+species, method="MC")
```

```
Testing diam _|_ height | species
Statistic (DEV):    2.026 df: NA p-value: 0.3735 method: MC
```

# 7 Log–linear models using the gRim package

Risk factors for coronary artery disease (CAD):

```
data(cad1)
head(cad1)
```

```
     Sex   AngPec        AMI QWave QWavecode    STcode STchange SuffHeartF
1   Male     None NotCertain    No    Usable    Usable       No         No
2   Male Atypical NotCertain    No    Usable    Usable       No         No
3 Female     None   Definite    No    Usable    Usable       No         No
4   Male     None NotCertain    No    Usable Nonusable       No         No
5   Male     None NotCertain    No    Usable Nonusable       No         No
6   Male     None NotCertain    No    Usable Nonusable       No         No
  Hypertrophi Hyperchol Smoker Inherit Heartfail CAD
1          No        No     No      No        No  No
2          No        No     No      No        No  No
3          No        No     No      No        No  No
4          No        No     No      No        No  No
5          No        No     No      No        No  No
6          No        No     No      No        No  No
```

The function **dmod()** is used for specifying log–linear models.

- Data must be a table or dataframe (which can be coerced to a table)
- Model given as generating class:
  - A right–hand–sided formula or
  - A list.
  - Variable names may be abbreviated:

```
mm <- dmod(~CAD:Sex+Sex:Smoker:Inherit, data=cad1)
mm <- dmod(list(c("CAD","Sex"), c("Sex","Smoker","Inherit")), data=cad1)
mm <- dmod(~C:Se+Se:Sm:I, data=cad1)
mm
```

```
Model: A dModel with 4 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :        1078.94 mdim :    9 aic :       1096.94
 ideviance :          21.71 idf  :    5 bic :       1128.11
 deviance  :          30.46 df   :    6
```

The generating class as a list is retrieved with **terms()** and as a formula with **formula()** :

```
str(terms(mm))
```

```
List of 2
 $ : chr [1:2] "CAD" "Sex"
 $ : chr [1:3] "Sex" "Smoker" "Inherit"
```

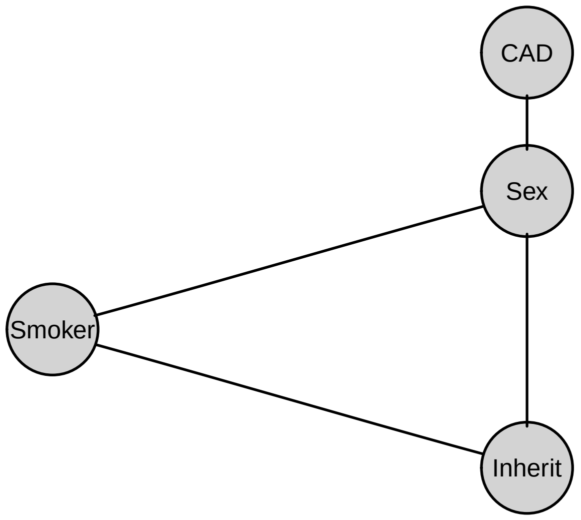
```
formula(mm)
```

```
~CAD * Sex + Sex * Smoker * Inherit
```

## 7.1 Plotting the dependence graph

If `Rgraphviz` is installed, graphs (and models) can be plotted with **plot()** . Otherwise **iplot()** may be used.

```
plot(mm)
```



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

```
# Default: a graphNEL object
DG <- ugList(terms(mm))
DG
```

```
A graphNEL graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

```
# Alternative: an adjacency matrix
ugList(terms(mm), result="matrix")
```

```
        CAD Sex Smoker Inherit
CAD       0   1      0       0
Sex       1   0      1       1
Smoker    0   1      0       1
Inherit   0   1      1       0
```

## 7.2 Model specification shortcuts

### Shortcuts for specifying some models

```
str(terms(dmod(~.^., data=cad1))) ## Saturated model
```

```
List of 1
 $ : chr [1:14] "Sex" "AngPec" "AMI" "QWave" ...
```

```
str(terms(dmod(~.^1, data=cad1))[1:4]) ## Independence model
```

```
List of 4
 $ : chr "Sex"
 $ : chr "AngPec"
 $ : chr "AMI"
 $ : chr "QWave"
```

```
str(terms(dmod(~.^3, data=cad1))[1:4]) ## All 3-factor model
```

```
List of 4
 $ : chr [1:3] "Sex" "AngPec" "AMI"
 $ : chr [1:3] "Sex" "AngPec" "QWave"
 $ : chr [1:3] "Sex" "AngPec" "QWavecode"
 $ : chr [1:3] "Sex" "AngPec" "STcode"
```

## Useful to combine with specification of a marginal table:

```
marg <- c("Sex", "Smoker", "Inherit")
str(terms(dmod(~.^., data=cad1, margin=marg))) ## Saturated model
```

```
List of 1
 $ : chr [1:3] "Sex" "Smoker" "Inherit"
```

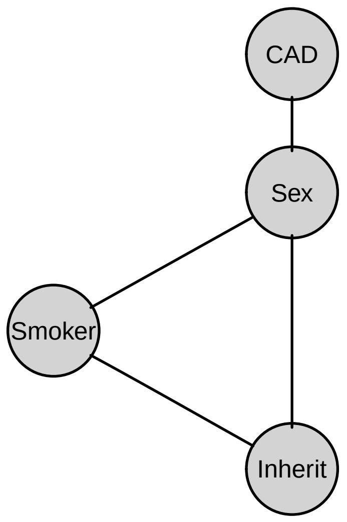
```
str(terms(dmod(~.^1, data=cad1, margin=marg))) ## Independence model
```

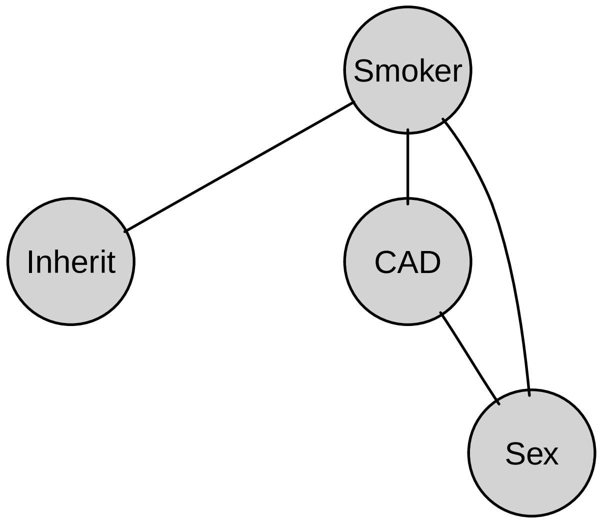
```
List of 3
 $ : chr "Sex"
 $ : chr "Smoker"
 $ : chr "Inherit"
```

## 7.3 Altering graphical models

Natural operations on graphical models: add and delete edges

```
mm <- dmod(~CAD:Sex+Sex:Smoker:Inherit, data=cad1)
mm2 <- update(mm, list(dedge=~Sex:Inherit, aedge=~CAD:Smoker)) # No abbreviations
par(mfrow=c(1,2)); plot(mm); plot(mm2)
```





## 7.4 Model comparison

Models are compared with **compareModels()** .

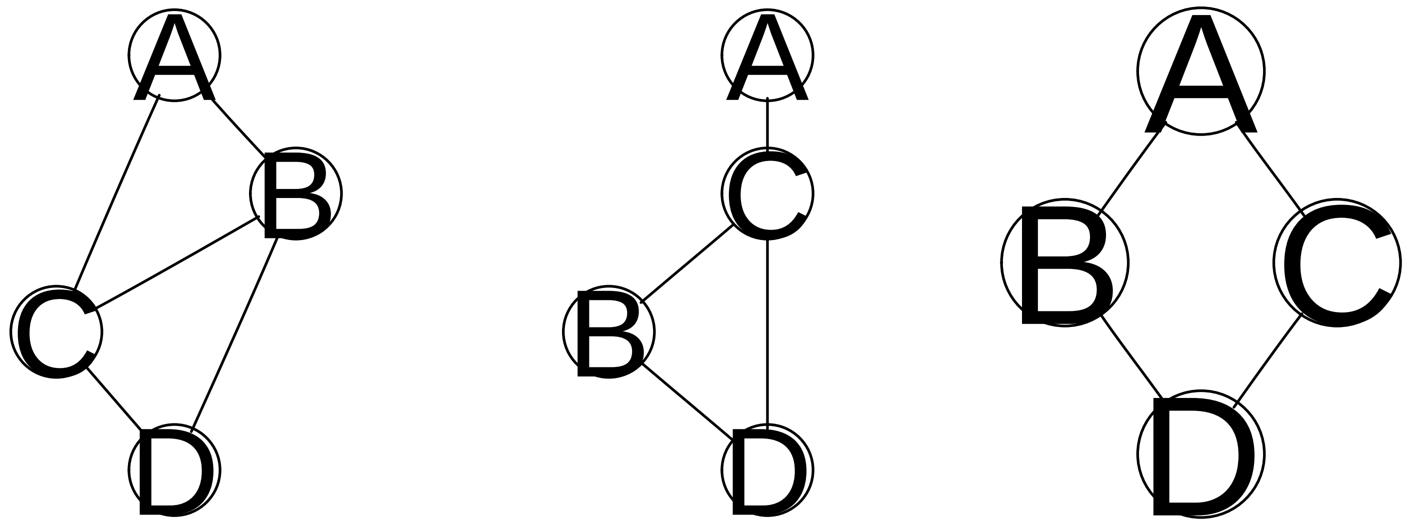
```
mm <- dmod(~CAD:Sex+Sex:Smoker:Inherit, data=cad1)
mm2 <- update(mm, list(dedge=~Sex:Inherit+CAD:Sex)) # No abbreviations
compareModels(mm,mm2)
```

```
Large:
  :"CAD" "Sex"
  :"Sex" "Smoker" "Inherit"
Small:
  :"Sex" "Smoker"
  :"Smoker" "Inherit"
  :"CAD"
-2logL:     8.84 df: 3 AIC(k= 2.0):     2.84 p.value: 0.219414
```

## 7.5 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.

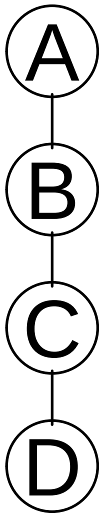
Result: The test for removal of $e = \{u, v\}$ which is contained in one clique $C$ only can be made as a test for $u \perp\!\!\!\perp v | C \setminus \{u, v\}$ in the $C$–marginal table.

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

## 7.6 Decomposable models – adding edges

More tricky when adding edge to a decomposable model

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



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)
```

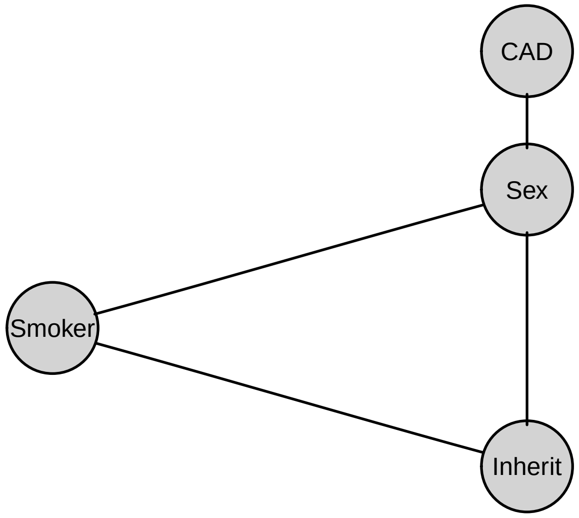
```
[1] "A" "B" "C" "D"
```

## 7.7 Test for adding and deleting edges

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

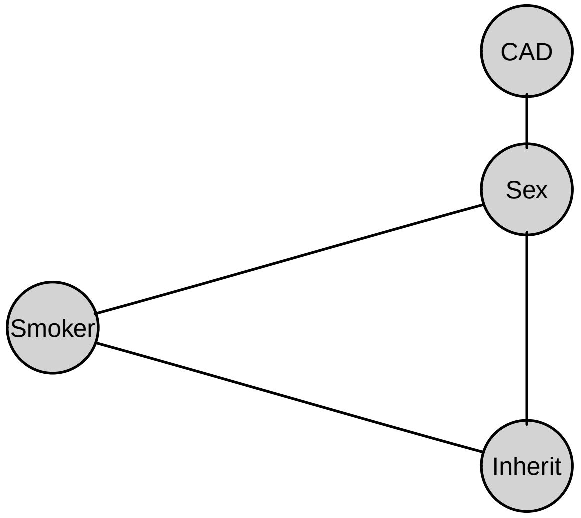
```
mm <- dmod(~CAD:Sex+Sex:Smoker:Inherit, data=cad1)
plot(mm)
testdelete(mm, edge=c("Sex","Smoker"))
```

```
dev:     6.083 df:  2 p.value: 0.04775 AIC(k=2.0):    2.1 edge: Sex:Smoker
host:  Sex Smoker Inherit
Notice: Test performed in saturated marginal model
```



```
mm <- dmod(~CAD:Sex+Sex:Smoker:Inherit, data=cad1)
plot(mm)
testadd(mm, edge=c("CAD","Smoker"))
```

```
dev:   12.972 df:  2 p.value: 0.00152 AIC(k=2.0):   -9.0 edge: CAD:Smoker
host:  Sex Smoker CAD
Notice: Test performed in saturated marginal model
```



## 7.8 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)

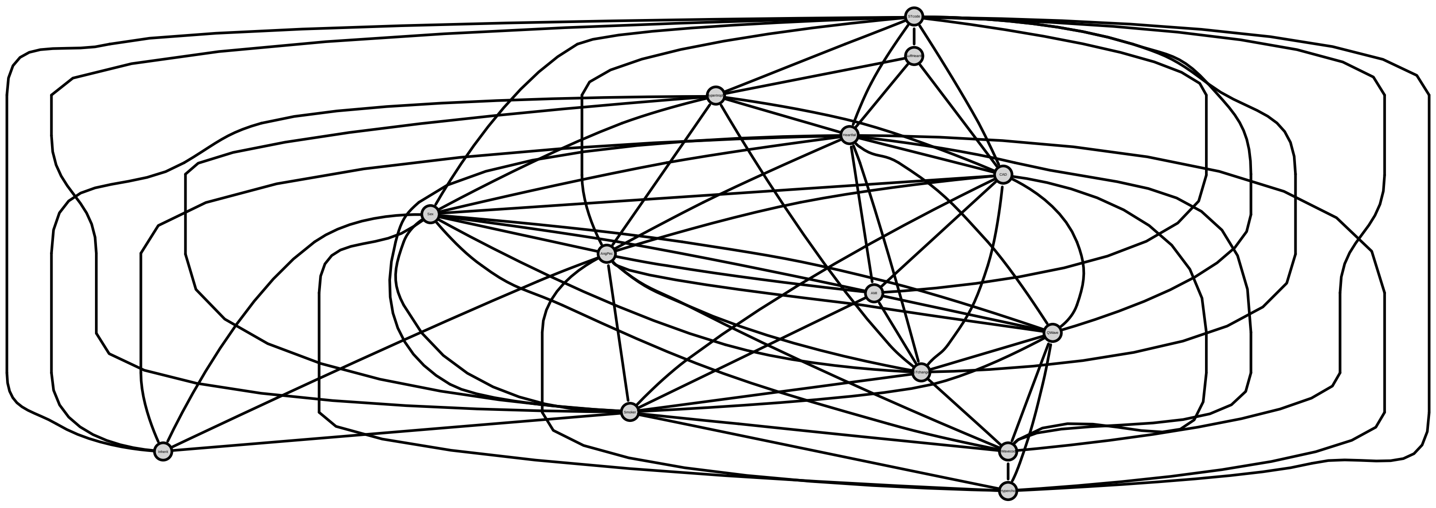
```
args(stepwise.iModel)
```

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

```
dm1 <- dmod(~.^., data=cad1)
dm2 <- stepwise(dm1, details=1)
```

```
STEPWISE:
 criterion: aic ( k = 2 )
 direction: backward
 type      : decomposable
 search    : all
 steps     : 1000
. BACKWARD: type=decomposable search=all, criterion=aic(2.00), alpha=0.00
. Initial model: is graphical=TRUE is decomposable=TRUE
  change.AIC   -7.2946 Edge deleted: Hyperchol SuffHeartF
  change.AIC   -9.9724 Edge deleted: SuffHeartF STchange
  change.AIC   -8.9869 Edge deleted: Hyperchol STchange
  change.AIC   -6.9429 Edge deleted: Hyperchol AMI
  change.AIC   -5.0911 Edge deleted: Inherit SuffHeartF
  change.AIC   -8.3854 Edge deleted: SuffHeartF AngPec
  change.AIC   -5.8491 Edge deleted: Inherit AMI
  change.AIC   -3.3619 Edge deleted: SuffHeartF AMI
  change.AIC   -4.7562 Edge deleted: SuffHeartF Sex
  change.AIC   -4.8269 Edge deleted: SuffHeartF Smoker
  change.AIC   -3.8263 Edge deleted: AMI Hypertrophi
  change.AIC   -3.1899 Edge deleted: STchange Inherit
  change.AIC   -2.6369 Edge deleted: CAD Hyperchol
  change.AIC   -7.6756 Edge deleted: Inherit CAD
  change.AIC   -2.1665 Edge deleted: SuffHeartF QWavecode
  change.AIC   -2.9359 Edge deleted: SuffHeartF QWave
  change.AIC   -1.7686 Edge deleted: QWavecode AMI
  change.AIC   -0.6979 Edge deleted: Inherit Hyperchol
  change.AIC   -2.5723 Edge deleted: Inherit QWavecode
  change.AIC   -1.5910 Edge deleted: Hyperchol Hypertrophi
  change.AIC   -3.7745 Edge deleted: Hypertrophi QWavecode
  change.AIC   -1.1525 Edge deleted: Inherit QWave
  change.AIC   -3.9161 Edge deleted: QWave Hypertrophi
```

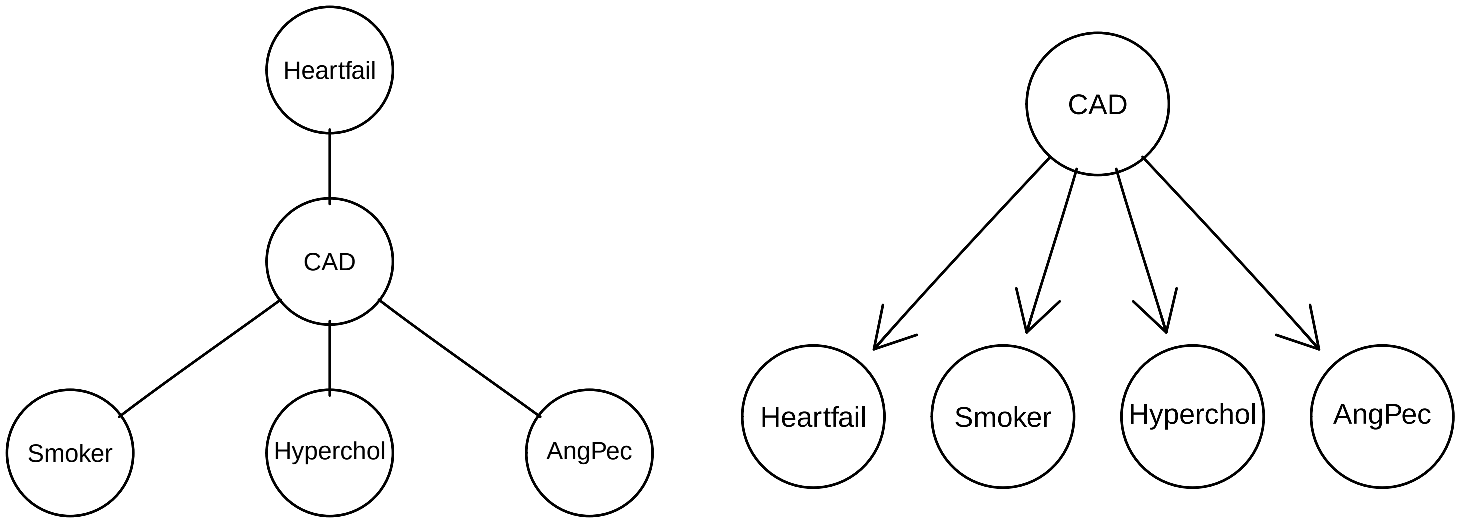
```
plot(dm2)
```



# 8 From graph and data to network

Consider naive Bayesian model for CAD data: All risk factors/symptoms are conditionally independent given the disease:

```
par(mfrow=c(1,2))
plot(UG  <- ug(~Heartfail:CAD+Smoker:CAD+Hyperchol:CAD+AngPec:CAD))
plot(DAG <- dag(~Heartfail:CAD+Smoker:CAD+Hyperchol:CAD+AngPec:CAD))
```



From a statistical point of view these two models are equivalent.

Given either a DAG or an UG and data (either as a table or as a dataframe) we can construct BN's on the fly:

```
cadmod1 <- compile(grain(UG, cad1))
cadmod2 <- compile(grain(DAG, cad1))
```

```
querygrain(cadmod1, nodes="CAD")
```

```
$CAD
CAD
       No       Yes
0.5466102 0.4533898
```

```
querygrain(cadmod2, nodes="CAD")
```

```
$CAD
CAD
       No       Yes
0.5466102 0.4533898
```

# 9 Prediction

We shall try to predict CAD in the validation dataset cad2

```
data(cad2)
head(cad2,3)
```

```
     Sex AngPec        AMI QWave QWavecode    STcode STchange SuffHeartF
1   Male   None NotCertain    No    Usable    Usable      Yes        Yes
2 Female   None NotCertain    No    Usable    Usable      Yes        Yes
3 Female   None NotCertain    No Nonusable Nonusable       No         No
  Hypertrophi Hyperchol Smoker Inherit Heartfail CAD
1          No        No   <NA>      No        No  No
2          No        No   <NA>      No        No  No
3          No       Yes   <NA>      No        No  No
```

using **predict.grain()** .

```
args(predict.grain)
```

```
function (object, response, predictors = setdiff(names(newdata),
    response), newdata, type = "class", ...)
```

```
NULL
```

```
(pred1 <- predict(cadmod1, resp="CAD", newdata=cad2[1:4,], type="class"))
```

```
$pred
$pred$CAD
[1] "No" "No" "No" "No"


$pFinding
[1] 0.13817620 0.13817620 0.11021122 0.04127698
```

```
(pred2 <- predict(cadmod1, resp="CAD", newdata=cad2[1:4,], type="dist"))
```

```
$pred
$pred$CAD
            No        Yes
[1,] 0.9135068 0.08649323
[2,] 0.9135068 0.08649323
[3,] 0.6786962 0.32130376
[4,] 0.6040489 0.39595114
```

```
$pFinding
[1] 0.13817620 0.13817620 0.11021122 0.04127698
```

## 10 Other things in `gRim`

`gRim` also implements

- Graphical Gaussian models (aka. covariance selection models)
- Homogeneous mixed graphical interaction models for continuous and discrete variables.

Estimation in log–linear and in Gaussian models is based on underlying C–code – fast!

Estimation in mixed models is based on R–code only – somewhat slower.

## 11 Built for speed, comfort or safety?

Set operations (book keeping) is an essential part of graphical modelling software; and this must be fast.

EXAMPLE: Find the unique elements in a vector:

```
x <- c('a','v','a','a','b','b','j','j','v','a','a','b','b','j','j')
M <- 50000
system.time({for (ii in 1:M) unique(x)})
```

```
   user  system elapsed
   0.37    0.00    0.38
```

```
system.time({for (ii in 1:M) unique.default(x)})
```

```
   user  system elapsed
   0.14    0.00    0.14
```

```
uniquePrim <- function (x) {
     x[!duplicated.default(x)]}
system.time({for (ii in 1:M) uniquePrim(x)})
```

```
   user  system elapsed
   0.12    0.00    0.13
```

EXAMPLE: Find the cliques of an undirected graph. Two options:

- **maxClique()** (from RBGL); works on graphNEL objects
- **maxCliqueMAT()** (from gRbase); works on adjacency matrices

```
maxClique <- function (g, nodes = NULL, edgeMat = NULL) {
   if (!missing(g) && isDirected(g))
     stop("only appropriate for undirected graphs")
   if (!(missing(g)) & (!is.null(nodes) | !is.null(edgeMat)))
     stop("if g is supplied, must not supply nodes or edgeMat")
   if (is.null(nodes))
     gn = g@nodes
   else gn = nodes
   if (is.null(edgeMat))
     em <- edgeMatrix(g)
   else em = edgeMat
   nv <- length(gn)
   ne <- ncol(em)
   ans <- .Call("maxClique", as.integer(nv), as.integer(ne),
                 as.integer(em - 1), PACKAGE = "RBGL")
   ans_names <- lapply(ans, function(x) {
     gn[x]
   })
   list(maxCliques = ans_names)
 }
```

```
maxCliqueMAT <- function (amat) {
   vn <- dimnames(amat)[[2L]]
   if (class(amat) == "dgCMatrix") {
     em <- t.default(sp_fromto(amat))
   }
   else {
     em <- t.default(sp_fromto(asdgCMatrix(amat)))
   }
   ans2 <- maxClique(nodes = vn, edgeMat = em)
   ans2
 }
```

Both functions a based on low–level C/Fortran code which does that real work – so we would expect similar performance wrt. time...

```
ff <- ~a:b+b:c+c:d+d:e+e:f+f:g+g:h+h:a # A cycle
ug.NEL <- ug(ff)
ug.MAT <- ug(ff, result="matrix")
```

```
M <- 1000
system.time({for (ii in 1:M) maxClique(ug.NEL)})
```

```
   user  system elapsed 
   0.49    0.00    0.49 
```

```
system.time({for (ii in 1:M) maxCliqueMAT(ug.MAT)})
```

```
   user  system elapsed
   0.10    0.00    0.09
```

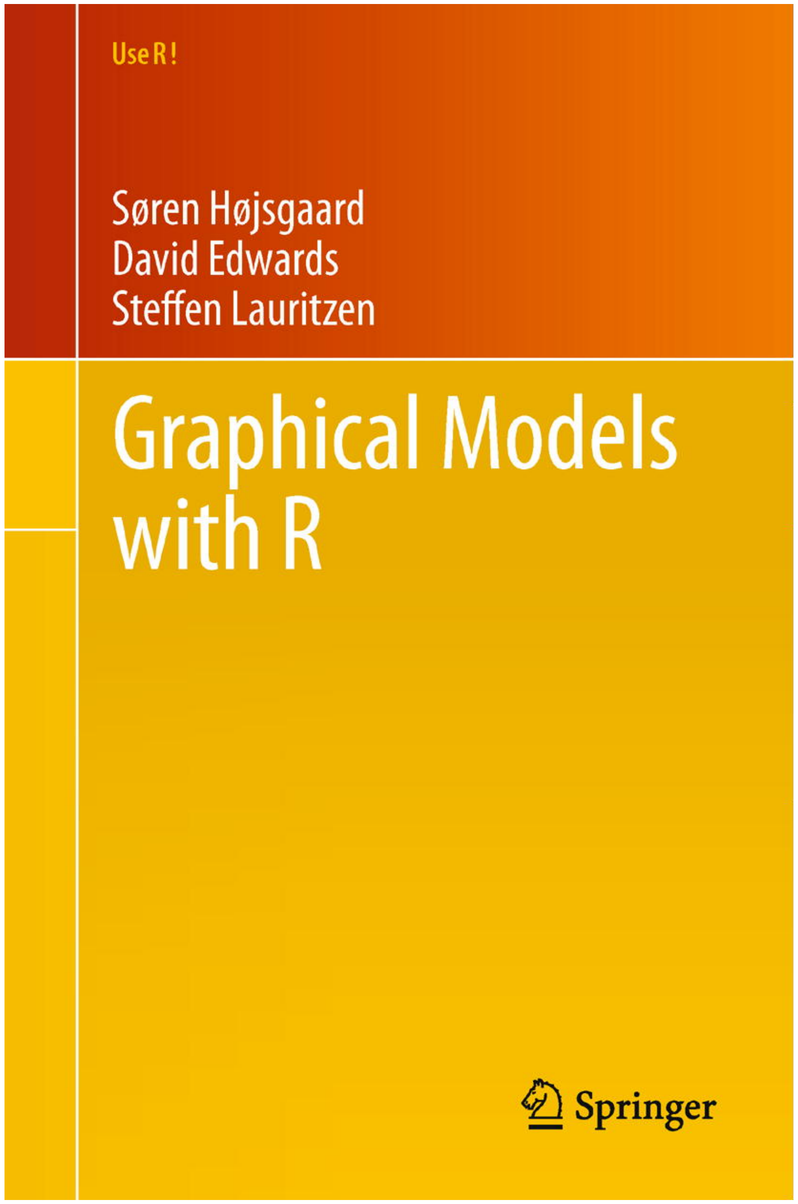
Much time is spent on the “overhead” associated with the S4 classes and related topics. (The good old S3 methods are considerably faster - IMHO).

## 12 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.

## 13 Book: Graphical Models with R


Use R!

Søren Højsgaard
David Edwards
Steffen Lauritzen

Graphical Models with R

Springer