Bayesian networks in R with the gRain package

Søren Højsgaard
Aalborg University, Denmark

gRain version 1.2-4 as of 2015-09-10

Contents

1 Introduction

The gRain package implements propagation in [gra]phical [i]ndependence [n]etworks (hereafter abbreviated grain). Such networks are also known as probabilistic networks and Bayesian networks.

To cite gRain in publications, please use:

> Søren Højsgaard (2012). Graphical Independence Networks with the gRain Package for R. Journal of Statistical Software, 46(10), 1-26. `http://www.jstatsoft.org/v46/i10/`.

and possibly also

> Søren Højsgaard, David Edwards and Steffen Lauritzen (2012). Graphical Models with R. Springer

More information about the package, other graphical modelling packages and development versions is available from

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

2 A worked example: chest clinic

This section reviews the chest clinic example of **?** (illustrated in Figure 1) and shows one way of specifying the model in `gRain`. **?** motivate the chest clinic example as follows:

"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."

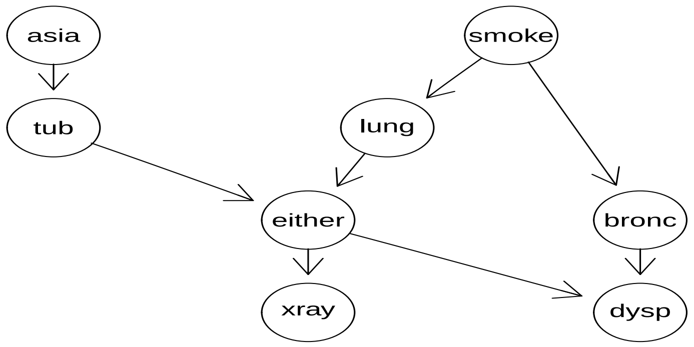


Figure 1: Chest clinic example from LS.

2.1 Building a network

A Bayesian network is a special case of graphical independence networks. In this section we outline how to build a Bayesian network. The starting point is a probability distribution factorising accoring to a DAG with nodes V. Each node $v \in V$ has a set $pa(v)$ of parents and each node $v \in V$ has a finite set of states. A joint distribution over the variables V can be given as

$$p(V) = \prod_{v \in V} p(v|pa(v)) \tag{1}$$

where $p(v|pa(v))$ is a function defined on $(v, pa(v))$. This function satisfies that $\sum_{v^*} p(v = v^*|pa(v)) = 1$, i.e. that for each configuration of the parents $pa(v)$, the sum over the levels of v equals one. Hence $p(v|pa(v))$ becomes the conditional distribution of v given $pa(v)$. In practice $p(v|pa(v))$ is specified as a table called a conditional probability table or a CPT for short. Thus, a Bayesian network can be regarded as a complex stochastic model built up by putting together simple components (conditional probability distributions).

Thus the DAG in Figure 1 dictates a factorization of the joint probability function as

$$p(V) = p(\alpha)p(\sigma)p(\tau|\alpha)p(\lambda|\sigma)p(\beta|\sigma)p(\epsilon|\tau,\lambda)p(\delta|\epsilon,\beta)p(\xi|\epsilon). \tag{2}$$

In (2) we have α = asia, σ = smoker, τ = tuberculosis, λ = lung cancer, β = bronchitis, ϵ = either tuberculosis or lung cancer, δ = dyspnoea and ξ = xray. Note that ϵ is a logical variable which is true if either τ or λ are true and false otherwise.

2.2 Queries to networks

Suppose we are given the evidence (sometimes also called "finding") that a set of variables $E \subset V$ have a specific value e^*. For example that a person has recently visited Asia and suffers from dyspnoea, i.e. $\alpha =$ yes and $\delta =$ yes.

With this evidence, we are often interested in the conditional distribution $p(v|E = e^*)$ for some of the variables $v \in V \setminus E$ or in $p(U|E = e^*)$ for a set $U \subset V \setminus E$.

In the chest clinic example, interest might be in $p(\lambda|e^*)$, $p(\tau|e^*)$ and $p(\beta|e^*)$, or possibly in the joint (conditional) distribution $p(\lambda, \tau, \beta|e^*)$.

Interest might also be in calculating the probability of a specific event, e.g. the probability of seeing a specific evidence, i.e. $p(E = e^*)$.

3 A one–minute version of gRain

3.1 Specifying a network

A simple way of specifying the model for the chest clinic example is as follows.

1. Specify conditional probability tables (with values as given in **?**):

```
> 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)
```

2. Compile list of conditional probability tables and create the network:

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

```
     asia
tub    yes   no
  yes 0.05 0.01
  no  0.95 0.99
```

```
> plist$either ## Notice: a logical node
```

```
, , tub = yes

      lung
either yes no
   yes   1  1
   no    0  0

, , tub = no

      lung
either yes no
   yes   1  0
   no    0  1

> net1 <- grain(plist)
> net1
Independence network: Compiled: FALSE Propagated: FALSE
  Nodes: chr [1:8] "asia" "tub" "smoke" "lung" "bronc" "either" "xray" ...
```

3.2 Querying a network

1. The network can be queried to give marginal probabilities:

```
> querygrain(net1, nodes=c("lung","bronc"), type="marginal")
$lung
lung
  yes    no
0.055 0.945

$bronc
bronc
 yes   no
0.45 0.55
```

2. Likewise, a joint distribution can be obtained:

```
> querygrain(net1,nodes=c("lung","bronc"), type="joint")
     bronc
lung     yes     no
  yes 0.0315 0.0235
  no  0.4185 0.5265
```

3. Evidence can be entered in one of these two equivalent forms:

```
> net12  <- setEvidence(net1, evidence=list(asia="yes", dysp="yes"))
> net12  <- setEvidence(net1,
+                        nodes=c("asia", "dysp"), states=c("yes", "yes"))
```

4. The probability of observing this evidence under the model is

```
> pEvidence( net12 )
[1] 0.004501375
```

5. The network can be queried again:

```
> querygrain( net12, nodes=c("lung","bronc") )
$lung
lung
       yes         no
0.09952515 0.90047485

$bronc
```

```
bronc
      yes        no
0.8114021 0.1885979
```

```
> querygrain( net12, nodes=c("lung","bronc"), type="joint" )
```

```
     bronc
lung         yes         no
  yes 0.06298076 0.03654439
  no  0.74842132 0.15205354
```

3.3 Conditioning on evidence with zero probability

Consider setting the evidence

```
> net13 <- setEvidence(net1,nodes=c("either", "tub"),
+                        states=c("no","yes"))
```

Under the model, this finding has zero probability;

```
> pEvidence( net13 )
```

```
[1] 0
```

Therefore, all conditional probabilities are (under the model) undefined;

```
> querygrain( net13, nodes=c("lung","bronc"), type="joint" )
```

```
     bronc
lung  yes  no
  yes NaN NaN
  no  NaN NaN
```

We can look closer into this zero–probability issue. Because the node `either` is logical, half of the configurations will have zero probability:

```
> tt <- querygrain( net1, type="joint")
> sum(tt==0)/length(tt)
```

```
[1] 0.5
```

In particular the configuration above has zero probability

```
> sum(tableSlice(tt, c("either","tub"), c("no","yes")))
```

```
[1] 0
```

Zero probailities (or almost zero probabilities) also arise in a different in a different setting. Consider this example

```
> yn <- c("yes","no")
> eps <- 1e-100
> a    <- cptable(~a,   values=c(1,eps),levels=yn)
> b.a  <- cptable(~b+a, values=c(1,eps,eps,1),levels=yn)
> c.b  <- cptable(~c+b, values=c(1,eps,eps,1),levels=yn)
> plist <- compileCPT(list(a, b.a, c.b))
> bn   <- grain(plist)
> ( tt   <- querygrain(bn, type="joint") )
```

```
, , c = yes

     b
a        yes     no
  yes  1e+00 1e-200
  no  1e-200 1e-200

, , c = no

     b
```

```
a        yes     no
  yes 1e-100 1e-100
  no  1e-300 1e-100
```

```
> querygrain(setEvidence(bn, nodes=c("a","c"), state=c("no", "yes")))
```

```
$a
a
yes  no 
  0   1 

$b
b
yes  no 
0.5 0.5 

$c
c
yes  no 
  1   0 
```

No problem so far, but if `eps` is made smaller numerical problems arise:

```
> eps <- 1e-200
> a    <- cptable(~a,   values=c(1,eps),levels=yn)
> b.a  <- cptable(~b+a, values=c(1,eps,eps,1),levels=yn)
> c.b  <- cptable(~c+b, values=c(1,eps,eps,1),levels=yn)
> plist <- compileCPT(list(a, b.a, c.b))
> bn   <- grain(plist)
> ( tt   <- querygrain(bn, type="joint") )
```

```
, , c = yes

     b
a     yes no
  yes   1  0
  no    0  0

, , c = no

     b
a        yes     no
  yes 1e-200 1e-200
  no   0e+00 1e-200
```

```
> querygrain(setEvidence(bn, nodes=c("a","c"), state=c("no", "yes")))
```

```
$a
a
yes  no 
NaN NaN 

$b
b
yes  no 
NaN NaN 

$c
c
yes  no 
NaN NaN 
```

4 Hard and virtual (likelihood) evidence

Below we describe how to work with virtual evidence (also known as likelihood evidence) in `gRain`. This is done via the function `setEvidence()`.

The clique potential representation in a Bayesian network gives

$$p(x) \propto \psi(x) = \prod_C \psi_C(x_C)$$

where we recall that the whole idea in computations with Bayesian networks is to avoid calculation the product on the right hand side. Instead computations are based on propagation (multiplying, dividing and summing clique potentials ψ_C in an appropriate order, and such an appropriate order comes from a junction tree). The normalizing constant, say $c = \sum_x \psi(x)$, comes out of propagation as a "by product".

Suppose a set of nodes E are known to have a specific value, i.e. $x_E = x_E^*$. This is called hard evidence. The probability of the event $x_E = x_E^*$ is

$$p(x_E = x_E^*) = E_p\{I(x_E = x_E^*)\} = \sum_x I(x_E = x_E^*)p(x) = \frac{1}{c}\sum_x I(x_E = x_E^*)\psi(x)$$

The computations are based on modifying the clique potentials ψ_C by giving value zero to states in ψ_C which are not consistent with $x_E = x_E^*$. This can be achieved with an indicator function, say $L_C(x_C)$ such that we obtain a set of new potentials $\tilde{\psi}_C = L_C(x_C)\psi_C(x_C)$. Propagation with these new potentials gives, as a by product, $\tilde{c} = \sum \tilde{\psi}(x)$ where $\tilde{\psi}(x) = \prod_C \tilde{\psi}_C(x_C)$. Consequently, we have $p(x_E = x_E^*) = \tilde{c}/c$.

In a more general setting we may have non–negative weights $L(x)$ for each value of x. We may calculate

$$E_p\{L(X)\} = \sum_x L(x)p(x)$$

If $L(X)$ factorizes as $L(X) = L_C(X_C)$ then the computations are carried out as outlined above, i.e. by the message passing scheme.

4.1 An excerpt of the chest clinic network

Consider the following excerpt of the chest clinic network which is described in the paper mentioned above.

```
> yn <- c("yes","no")
> a    <- cptable(~asia, values=c(1,99),levels=yn)
> t.a  <- cptable(~tub|asia, values=c(5,95,1,99),levels=yn)
> ( plist1 <- compileCPT( list( a, t.a ) ) )
```

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

```
> plist1[[1]]
```

```
asia
 yes   no
0.01 0.99
```

```
> plist1[[2]]
```

```
     asia
tub    yes   no
  yes 0.05 0.01
  no  0.95 0.99
```

```
> ( chest1 <- grain(plist1) )

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

> querygrain( chest1 )

$asia
asia
 yes   no
0.01 0.99

$tub
tub
   yes     no
0.0104 0.9896
```

4.2 Specifying hard evidence

Suppose we want to make a diagnosis about tuberculosis given the evidence that a person has recently been to Asia. The functions `setFinding()` (which has been in `gRain` for years) and `setEvidence()` (which is a recent addition to `gRain`) can both be used for this purpose. The following forms are equivalent.

```
> setFinding(  chest1, nodes="asia", states="yes")

Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:2] "asia" "tub"
  Evidence:
  nodes is.hard.evidence hard.state
1  asia             TRUE        yes
  pEvidence: 0.010000

> setEvidence( chest1, nodes="asia", states="yes")

Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:2] "asia" "tub"
  Evidence:
  nodes is.hard.evidence hard.state
1  asia             TRUE        yes
  pEvidence: 0.010000

> setEvidence( chest1, evidence=list(asia="yes"))

Independence network: Compiled: TRUE Propagated: TRUE
  Nodes: chr [1:2] "asia" "tub"
  Evidence:
  nodes is.hard.evidence hard.state
1  asia             TRUE        yes
  pEvidence: 0.010000

> querygrain( setEvidence( chest1, evidence=list(asia="yes")) )

$asia
asia
yes  no
  1   0

$tub
tub
 yes   no
0.05 0.95
```

4.3 What is virtual evidence (also called likelihood evidence) ?

Suppose we do not know with certainty whether a patient has recently been to Asia (perhaps the patient is too ill to tell). However the patient (if he/she is Caucasian) may be unusually tanned and this lends support to the hypothesis of a recent visit to Asia.

To accommodate we create an extended network with an extra node for which we enter evidence. However, it is NOT necessary to do so in practice, because we may equivalently enter the virtual evidence in the original network.

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

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

```
          asia
guess.asia yes  no
       yes 0.8 0.1
       no  0.2 0.9
```

This reflects the assumption that for patients who have recently been to Asia we would guess so in 80% of the cases, whereas for patients who have not recently been to A we would (erroneously) guess that they have recently been to Asia in 10% of the cases.

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

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

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

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

```
> querygrain( chest2 )
```

```
$asia
asia
 yes   no
0.01 0.99

$tub
tub
   yes     no
0.0104 0.9896

$guess.asia
guess.asia
  yes    no
0.107 0.893
```

Now specify the guess or judgment, that the person has recently been to Asia:

```
> querygrain( setEvidence( chest2, evidence=list(guess.asia="yes")) )
```

```
$asia
asia
       yes         no
0.07476636 0.92523364

$tub
tub
       yes         no
0.01299065 0.98700935
```

```
$guess.asia
guess.asia
yes  no
  1   0
```

4.4 Specifying virtual evidence

The same guess or judgment can be specified as virtual evidence (also called likelihood evidence) for the original network:

```
> querygrain( setEvidence( chest1, evidence=list(asia=c(.8, .1))) )
```

```
$asia
asia
       yes         no
0.07476636 0.92523364

$tub
tub
       yes         no
0.01299065 0.98700935
```

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

```
> querygrain( setEvidence( chest1, evidence=list(asia=c(1, 0))) )
```

```
$asia
asia
yes  no
  1   0

$tub
tub
 yes   no
0.05 0.95
```

4.5 A mixture of a discrete and a continuous variable

`grain` only handles discrete variables with a finite state space, but using likelihood evidence it is possible to work with networks with both discrete and continuous variables (or other types of variables). This is possible only when he networks have a specific structure. This is possible when no discrete variable has non–discrete parents.

Take a simple example: x is a discrete variable with levels 1 and 2; $y_1|x = k \sim N(\mu_k, \sigma_k^2)$ and $y_2|x = k \sim Poi(\lambda_k)$ where $k = 1, 2$. The joint distribution is

$$p(x, y_1, y_2) = p(x)p(y_1|x)p(y_2|x)$$

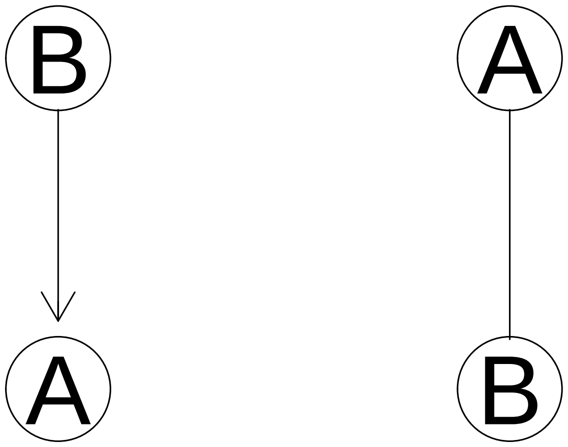
Suppose the interest is in the distribution of x given $y_1 = y_1^*$ and $y_2 = y_2^*$. We then have

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

5 Building networks from data

The following two graphs specify the same model:

```
> dG  <- dag(~A:B)
> uG  <- ug(~A:B)
> par(mfrow=c(1,2)); plot( dG ); plot( uG )
```



Suppose data is

```
> dat <-as.table(parray(c("A","B"), levels=c(2,2), values=c(0,0,2,3)))
    B
A    B1 B2
  A1  0  2
  A2  0  3
> class( dat )
[1] "table"  "parray" "array"
```

A network can be built from data using:

```
> gr.dG <- compile( grain( dG, dat ) )
NAs found in conditional probability table(s) for nodes: A
  ... consider using the smooth argument
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:2] "A" "B"
> gr.uG <- compile( grain( uG, dat ) )
Independence network: Compiled: TRUE Propagated: FALSE
  Nodes: chr [1:2] "A" "B"
```

However, when there are zeros in the table, care must be taken.

5.1 Extracting information from tables

In the process of creating networks, conditional probability tables are extracted when the graph is a dag and clique potentials are extracted when the graph is a chordal (i.e. triangulated) undirected graph. This takes place as follows (internally):

```
> extractCPT( dat, dG )
NAs found in conditional probability table(s) for nodes: A
  ... consider using the smooth argument
$A
```

```
    B
A      B1  B2
  A1 NaN 0.4
  A2 NaN 0.6

$B
B
B1 B2
 0  1

attr(,"class")
[1] "extractCPT" "list"
```

```
> c( extractPOT( dat, uG ) )
```

```
[[1]]
    B
A     B1  B2
  A1   0 0.4
  A2   0 0.6
```

The conditional probability table $P(A|B)$ contains `NaN`s because

$$P(A|B = B1) = \frac{n(A, B = B1)}{\sum_A n(A, B = B1)} = \frac{0}{0} = \text{NaN}$$

For this reason the network `gr.dG` above will fail to compile whereas `gr.uG` will work, but it may not give the expected results.

5.2 Using smooth

To illustrate what goes on, we can extract the distributions from data as follows:

```
> p.A.g.B <- tableDiv(dat, tableMargin(dat, "B"))
```

```
    A
B      A1  A2
  B1 0.0 0.0
  B2 0.4 0.6
```

```
> p.B <- tableMargin(dat, "B")/sum(dat)
```

```
B
B1 B2
 0  1
```

```
> p.AB <- tableMult( p.A.g.B, p.B)
```

```
    A
B      A1  A2
  B1 0.0 0.0
  B2 0.4 0.6
```

However, the result is slightly misleading because `tableDiv` sets $0/0 = 0$.

In `grain` there is a `smooth` argument that will add a small number to the cell entries before extracting tables, i.e.

$$P(A|B = B1) = \frac{n(A, B = B1) + \epsilon}{\sum_A (n(A, B = B1) + \epsilon)} = \frac{\epsilon}{2\epsilon} = 0.5$$

and

$$P(B) = \frac{\sum_A (n(A, B) + \epsilon)}{\sum_{AB} (n(A, B) + \epsilon)}$$

We can mimic this as follows:

```
> e <- 1e-2
> (dat.e <- dat + e)
    B
A      B1   B2
  A1 0.01 2.01
  A2 0.01 3.01
```

```
> pe.A.g.B <- tableDiv(dat.e, tableMargin(dat, "B"))
    A
B       A1    A2
  B1 0.000 0.000
  B2 0.402 0.602
```

```
> pe.B <- tableMargin(dat.e, "B")/sum(dat.e)
B
         B1          B2 
0.003968254 0.996031746 
```

```
> pe.AB  <- tableMult( pe.A.g.B, pe.B )
    A
B           A1        A2
  B1 0.0000000 0.0000000
  B2 0.4004048 0.5996111
```

However this resulting joint distribution is different from what is obtained from the adjusted table itself

```
> dat.e / sum( dat.e )
    B
A             B1          B2
  A1 0.001984127 0.398809524
  A2 0.001984127 0.597222222
```

This difference appears in the `grain` networks.

5.3 Extracting tables

One can do

```
> gr.dG <- compile( grain( dG, dat, smooth=e ) )
```

which (internally) corresponds to

```
> extractCPT( dat, dG, smooth=e)
$A
    B
A     B1        B2
  A1 0.5 0.4003984
  A2 0.5 0.5996016

$B
B
         B1          B2 
0.001992032 0.998007968 

attr(,"class")
[1] "extractCPT" "list"
```

We get

```
> querygrain( gr.dG )
```

```
$A
A
       A1        A2
0.4005968 0.5994032

$B
B
         B1          B2
0.001992032 0.998007968
```

```
> querygrain( gr.uG )
```

```
$A
A
 A1  A2
0.4 0.6

$B
B
B1 B2
 0  1
```

However, if we condition on `B=B1` we get:

```
> querygrain(setFinding(gr.dG, nodes="B", states="B1"))
```

```
$A
A
 A1  A2
0.5 0.5

$B
B
B1 B2
 1  0
```

```
> querygrain(setFinding(gr.uG, nodes="B", states="B1"))
```

```
$A
A
 A1  A2
NaN NaN

$B
B
 B1  B2
NaN NaN
```

so the “problem” with zero entries shows up in a different place. However, the answer is not necessarily wrong; the answer simply states that $P(A|B = B1)$ is undefined. To “remedy” we can use the `smooth` argument:

```
> gr.uG <- compile( grain( uG, dat, smooth=e) )
```

which (internally) corresponds to

```
> c( extractPOT( dat, uG, smooth=e ) )
```

```
[[1]]
    B
A             B1        B2
  A1 0.001984127 0.3988095
  A2 0.001984127 0.5972222
```

Notice that the results are not exactly identical:

```
> querygrain( gr.uG )
```

```
$A
A
       A1        A2 
0.4007937 0.5992063 

$B
B
         B1          B2 
0.003968254 0.996031746 

> querygrain( gr.dG )

$A
A
       A1        A2 
0.4005968 0.5994032 

$B
B
         B1          B2 
0.001992032 0.998007968 

> querygrain( setFinding(gr.uG, nodes="B", states="B1") )

$A
A
 A1  A2 
0.5 0.5 

$B
B
B1 B2 
 1  0 

> querygrain( setFinding(gr.dG, nodes="B", states="B1") )

$A
A
 A1  A2 
0.5 0.5 

$B
B
B1 B2 
 1  0 
```