gRain – [gRa]phical [i]ndependence [n]etworks in R

Søren Højsgaard [*]

March 7, 2008

Contents

[*]Institute of Genetics and Biotechnology, Aarhus University, Research Center Foulum, DK–8830 Tjele, Denmark

1 Introduction

The `gRain` package is an `R` package, (R Development Core Team 2007) for efficient calculation of (conditional) probability distributions in graphical independence networks, hereafter denoted `iNets`. Such independence networks are sometimes also denoted probabilistic expert systems. A special case of such networks is Bayesian networks.

The networks are restricted to consisting of discrete variables, each with a finite state space. The networks will typically satisfy conditional independence restrictions which enables the computations to be made very efficiently.

The `gRain` package is in its functionality similar to the `GRAPPA` suite of functions, (Green 2005) although there are important differences. The package implements the propagation algorithm of Lauritzen and Spiegelhalter (1988). For brevity we refer to Lauritzen and Spiegelhalter (1988) as LS.

2 A worked example: chest clinic

This section reviews the chest clinic example of LS (illustrated in Figure 1) and shows one way of specifying the model in `gRain`. Details of the steps will be given in later sections. Other ways of specifying a `iNet` are described in Section 8. LS 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.”

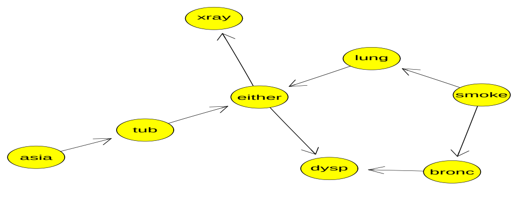

Figure 1: Chest clinic example from LS.

2.1 Building a `iNet`

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

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

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

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

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 `iNets`

Suppose we are given evidence 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. α = yes and δ = 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. $p(E = e^*)$.

2.3 A one–minute version of `gRain`

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 Lauritzen and Spiegelhalter (1988)):

```
> yn <- c("yes", "no")
> a <- cpt(~asia, values = c(1, 99), levels = yn)
> t.a <- cpt(~tub + asia, values = c(5, 95, 1, 99), levels = yn)
> s <- cpt(~smoke, values = c(5, 5), levels = yn)
> l.s <- cpt(~lung + smoke, values = c(1, 9, 1, 99), levels = yn)
> b.s <- cpt(~bronc + smoke, values = c(6, 4, 3, 7), levels = yn)
> e.lt <- cpt(~either + lung + tub, values = c(1, 0, 1, 0, 1, 0, 0,
+     1), levels = yn)
> x.e <- cpt(~xray + either, values = c(98, 2, 5, 95), levels = yn)
> d.be <- cpt(~dysp + bronc + either, values = c(9, 1, 7, 3, 8, 2,
+     1, 9), levels = yn)
```

2. Create the iNet from the conditional probability tables:

```
> plist <- cptspec(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
> in1 <- newgmInstance(plist)
> in1

Independence network: Compiled: FALSE Propagated: FALSE
```

3. The iNet can be queried to give marginal probabilities:

```
> querygm(in1, nodes = c("lung", "bronc"), type = "marginal")

$lung
lung
  yes    no
0.055 0.945

$bronc
bronc
 yes   no
0.45 0.55
```

   Likewise, a joint distribution can be obtained.

```
> querygm(in1, nodes = c("lung", "bronc"), type = "joint")

     bronc
lung     yes     no
  yes 0.0315 0.0235
  no  0.4185 0.5265
```

4. Evidence can be entered as:

```
> in12 <- enterEvidence(in1, nodes = c("asia", "dysp"), states = c("yes",
+     "yes"))
```

5. The iNet can be queried again:

```
> querygm(in12, nodes = c("lung", "bronc"))

$lung
lung
       yes         no
0.09952515 0.90047485

$bronc
bronc
      yes        no
0.8114021 0.1885979

> querygm(in12, nodes = c("lung", "bronc"), type = "joint")

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

3 Building and using iNets

3.1 Compilation and propagation

Before queries can be made to a iNet the iNet must be compiled (see Section B.1.1) and propagated (see Section B.1.2). These two steps are forced by the querygm function if necessary, but it is in some cases advantegous to do them explicitly.

3.1.1 Compilation of an `iNet`

Put briefly, compilation of an `iNet` involves the following steps: It is first checked whether the list of CPTs defines a directed acyclic graph DAG. If so, this dag is created; it is moralized and triangulated. The CPTs are transformed into potentials defined on the cliques of the triangulated graph. See Section B.1.1 for further details. The triangulated graph together with the corresponding clique potentials constitute an `iNet`. Thus the list of CPTs is merely one way of constructing an `iNet`. Consider again Bayesian network of Section 2.3:

```
> in1

Independence network: Compiled: FALSE Propagated: FALSE

> class(in1)

[1] "cpt-gmInstance" "gmInstance"
```

The class attributes show that the `iNet` derives from a list of CPTs. In Section **??** other ways of constructing an `iNet` are described.

```
> in1c <- compilegm(in1)

Independence network: Compiled: TRUE Propagated: FALSE

> class(in1c)

[1] "compgmInstance" "cpt-gmInstance" "gmInstance"
```

To be able to answer queries the `iNet` must be propagated which means that the clique potentials must be adjusted to each other in a specific way. See Section B.1.2 for details.

Default is that propagation are not carried out in connected with compilation but this can be changed by setting `propagate="TRUE"` in `compilegm()`

3.1.2 Propagation of an `iNet`

A compiled `iNet` can be propagated as follows. Note that there are various options to choose in this connection; see the documentation of `gRain` for details:

```
> in1c <- propagate(in1c)

Independence network: Compiled: TRUE Propagated: TRUE
```

3.2 Queries and evidence

3.2.1 Queries

As illustrated in Section 2.3, queries can be made to a `iNet` using the `querygm()` function. The result is by default an array (or a list of array(s)). Setting `return="data.frame"` causes the result to be returned as a dataframe (or a list of dataframes):

```
> querygm(in1c, nodes = c("lung", "bronc"), return = "data.frame")

$lung
    lung  Freq
yes  yes 0.055
no    no 0.945

$bronc
    bronc Freq
yes   yes 0.45
no     no 0.55
```

```
> querygm(in1c, nodes = c("lung", "bronc"), type = "joint", return = "data.frame")

  lung bronc   Freq
1  yes   yes 0.0315
2   no   yes 0.4185
3  yes    no 0.0235
4   no    no 0.5265
```

With `type="marginal"` the we get $P(\lambda)$ and $P(\beta)$. Setting `type="joint"` gives $P(\lambda, \beta)$.

Setting `type="conditional"` gives $P(\lambda|\beta)$, i.e. the distribution of the first variable in `nodes` given the remaining ones:

```
> querygm(in1c, nodes = c("lung", "bronc"), type = "conditional",
+     return = "data.frame")

  lung bronc       Freq
1  yes   yes 0.07000000
2   no   yes 0.93000000
3  yes    no 0.04272727
4   no    no 0.95727273
```

Omitting `nodes` implies that all nodes are considered.

3.2.2 Entering evidence

Suppose we want to enter the evidence that a person has recently been to Asia and suffers from dyspnoea. This can be done in one of two ways:

```
> in1c2 <- enterEvidence(in1c, nodes = c("asia", "dysp"), states = c("yes",
+     "yes"))
> in1c2 <- enterEvidence(in1c, evlist = list(c("asia", "yes"), c("dysp",
+     "yes")))
```

The evidence itself is displayed with:

```
> evidence(in1c2)

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

The probability of observing the evidence is:

```
> pevidence(in1c2)

[1] 0.004501375
```

The marginal, joint and conditional (conditional) probabilities are now:

```
> querygm(in1c2, nodes = c("lung", "bronc"))

$lung
lung
       yes         no
0.09952515 0.90047485

$bronc
bronc
      yes        no
0.8114021 0.1885979

> querygm(in1c2, nodes = c("lung", "bronc"), type = "joint")
```

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

```
> querygm(in1c2, nodes = c("lung", "bronc"), type = "conditional")
```

```
     bronc
lung         yes        no
  yes 0.07761966 0.1937688
  no  0.92238034 0.8062312
```

Note that the latter result is the conditional distribution of `lung` given `bronc` – but also conditional on the evidence.

3.2.3 Incremental specification of evidence

Evidence can be entered incrementally by calling `enterEvidence()` repeatedly. If doing so, it is advantagous to set `propagate=FALSE` in `enterEvidence()` and then only call the `propagate()` function at the end.

3.2.4 Retracting evidence

Evidence can be retracted (removed from the `iNet`) with

```
> in1c3 <- retractEvidence(in1c2, nodes = "asia")
> evidence(in1c3)
```

```
Evidence:
     variable state
[1,] dysp     yes
Pr(Evidence)= 0.004501375
```

Omitting `nodes` implies that all evidence is retracted, i.e. that the `iNet` is reset to its original status.

3.3 Miscellaneous

Summary Summaries of `iNet`s are can be obtained:

```
> summary(in1)
```

```
Nodes : asia tub smoke lung bronc either xray dysp
Compiled: FALSE Propagated: FALSE
```

```
> summary(in1c)
```

```
Nodes : asia tub smoke lung bronc either xray dysp
Compiled: TRUE Propagated: TRUE
 Number of cliques: 6
 Maximal clique size: 3
 Maximal number of configurations in cliques: NA
```

The `summary()` function can be a `type` argument. Possible values for `type` include `"rip"`, `"cliques"`, `"configurations"`.

Graphics The DAG in Figure 1 is obtained with `plot(pn)`, while the triangulated indirected graph in Figure 2 is obtained with `plot(pnc)`.

Odds and ends The functions `nodeNames` and `nodeStates` returns the nodes and their states. A potential can be turned into a dataframe or a numerical variables with `as.data.frame` and `as.numeric`.[1]

Internally in `gRain`, a CPT is internally represented as a `ctab` object, see the package documentation for details.

4 Fast computation of a joint distribution

If interest is in fast computation of the latter joint distribution one can force these variables to be in the same clique of the TUG as:

```
> in1c2 <- compilegm(in1, root = c("lung", "bronc", "tub"), propagate = TRUE)
```

Now compare the computing time of the of the objects, the second one being much faster:

```
> system.time({
+     for (i in 1:50) querygm(in1c, nodes = c("lung", "bronc", "tub"),
+         type = "joint")
+ })

   user  system elapsed
   5.55    0.01    5.57

> system.time({
+     for (i in 1:50) querygm(in1c2, nodes = c("lung", "bronc", "tub"),
+         type = "joint")
+ })

   user  system elapsed
   0.05    0.00    0.04
```

5 Simulation

It is possible to simulate data from an `iNet`. This uses the current clique, and thus generates values conditional on all evidence entered in tne `iNet`.

```
> simulate(in1c, nsim = 5)

  asia tub smoke lung bronc either xray dysp
1   no  no   yes   no   yes     no   no  yes
2   no  no    no   no   yes     no   no  yes
3   no  no    no   no    no     no   no   no
4   no  no   yes   no    no     no   no   no
5   no  no   yes   no    no     no   no   no
```

6 Prediction

A `predict` method is available for `iNets` for predicting a set of "responses" from a set of "explanatory variables". Two types of predictions can be made. The default is `type="class"` which assigns the value to the class with the highest probability:

```
> mydata

  bronc dysp either lung tub asia xray smoke
1   yes  yes    yes  yes  no   no  yes   yes
2   yes  yes    yes  yes  no   no  yes    no
3   yes  yes    yes   no yes   no  yes   yes
4   yes  yes     no   no  no  yes  yes    no
```

[1]SHD: Rewrite this part...

```
> predict(in1c, response = c("lung", "bronc"), newdata = mydata, predictors = c("smoke",
+     "asia", "tub", "dysp", "xray"), type = "class")

$pred
$pred$lung
[1] "yes" "no"  "no"  "no"

$pred$bronc
[1] "yes" "yes" "yes" "yes"


$pevidence
[1] 0.0508475880 0.0111697096 0.0039778200 0.0001082667
```

The output should be read carefully: Conditional on the first observation in `mydata`, the most probable value of `lung` is `"yes"` and the same is the case for `bronc`. This is not in general the same as saying that the most likely configuration of the two variables `lung` and `bronc` is `"yes"`.

Alternatively, one can obtain the entire conditional distribution:

```
> predict(in1c, response = c("lung", "bronc"), newdata = mydata, predictors = c("smoke",
+     "asia", "tub", "dysp", "xray"), type = "dist")

$pred
$pred$lung
           yes        no
[1,] 0.7744796 0.2255204
[2,] 0.3267670 0.6732330
[3,] 0.1000000 0.9000000
[4,] 0.3267670 0.6732330

$pred$bronc
           yes        no
[1,] 0.7181958 0.2818042
[2,] 0.6373009 0.3626991
[3,] 0.6585366 0.3414634
[4,] 0.6373009 0.3626991


$pevidence
[1] 0.0508475880 0.0111697096 0.0039778200 0.0001082667
```

7 Alternative ways of specifying an `iNet`

This section illustrates alternative ways of specifying an `iNet`.

7.1 Defining variables and states – a gmData object

We will in the following make use of a `gmData` object (as introduced by Dethlefsen and Højsgaard (2005)) for holding the specification of the variables in the `iNet`. Briefly, a `gmData` object is a *graphical meta data* object which is an abstraction of data types such as dataframes and tables.

A `gmData` object needs not contain any real data; it can simply be a specification of variable names and their corresponding levels (and several other characterstics, for example wheter a categorical variable should be regarded as being ordinal or nominal).

For the chest clinic example in Section 2 we build the `gmData` object as

```
> chestNames <- c("asia", "smoke", "tub", "lung", "bronc", "either",
+     "xray", "dysp")
```

```
> gmd <- newgmData(chestNames, valueLabels = c("yes", "no"))
> gmd

       varNames shortNames varTypes nLevels
asia       asia          a Discrete       2
smoke     smoke          s Discrete       2
tub         tub          t Discrete       2
lung       lung          l Discrete       2
bronc     bronc          b Discrete       2
either   either          e Discrete       2
xray       xray          x Discrete       2
dysp       dysp          d Discrete       2
To see the values of the factors use the 'valueLabels' function
```

7.2 Specification of conditional probabilities

The CPTs can be created with reference to the `gmData` object as follows:

```
> a <- cpt(~asia, values = c(1, 99), gmData = gmd)
> t.a <- cpt(~tub + asia, values = c(5, 95, 1, 99), gmData = gmd)
> s <- cpt(~smoke, values = c(5, 5), gmData = gmd)
> l.s <- cpt(~lung + smoke, values = c(1, 9, 1, 99), gmData = gmd)
> b.s <- cpt(~bronc + smoke, values = c(6, 4, 3, 7), gmData = gmd)
> e.lt <- cpt(~either + lung + tub, values = c(1, 0, 1, 0, 1, 0, 0,
+     1), gmData = gmd)
> x.e <- cpt(~xray + either, values = c(98, 2, 5, 95), gmData = gmd)
> d.be <- cpt(~dysp + bronc + either, values = c(9, 1, 7, 3, 8, 2,
+     1, 9), gmData = gmd)
```

Note: Instead of using formulae as in `~tub+asia` we can write e.g. `c("tub","asia")`.

7.3 Building the `iNet`

From a list of conditional probabilities and a corresponding `gmData` object we can build a `iNet` as above:

```
> plist <- cptspec(list(a, t.a, s, l.s, b.s, e.lt, x.e, d.be))
> in1 <- newgmInstance(plist, gmData = gmd)
```

8 Building a `iNet` from data

An `iNet` can be built from data in two different ways. Suppose we have data in the form of a dataframe of cases e.g. as generated by `simulate` in Section 5. We convert data into a table and the table into a `gmData` object:

```
> chestSim <- simulate(in1c, nsim = 1000)
> gcs <- as.gmData(xtabs(~., chestSim))
```

8.1 From a directed acyclic graph

The directed graph in Figure 1 can be specified as:

```
> g <- list(~asia, ~tub + asia, ~smoke, ~lung + smoke, ~bronc + smoke,
+     ~either + lung + tub, ~xray + either, ~dysp + bronc + either)
> dag <- newdagsh(g)
```

An `iNet` can be built from the graph and the `gmData` object. In this process, the CPTs are estimated from data in `chestSim` as the relative frequencies. To avoid zeros in the CPTs one can choose to add a small number, e.g. `smooth=0.1` to all entries which are zero in the data:

```
> in1x <- newgmInstance(dag, gmData = gcs)
> in1x <- compilegm(in1x, propagate = TRUE, smooth = 0.1)
```

8.2 From a triangulated undirected graph

Alternatively, an `iNet` can be built from an undirected (but triangualted) graph. The undirected graph in Figure 2 can be specified as:

```
> g <- list(~asia + tub, ~either + lung + tub, ~either + lung + smoke,
+     ~bronc + either + smoke, ~bronc + dysp + either, ~either + xray)
> ug <- newugsh(g)
```

An `iNet` can be built from the graph and the `gmData` object. In this process, the clique potentials are estimated as the respective frequencies in the data:

```
> in1y <- newgmInstance(ug, gmData = gcs)
> in1y <- compilegm(in1y, propagate = TRUE)
```

9 Discussion and perspectives

10 Acknowledgements

Thanks to Peter J. Green for providing the R and Fortran code for the Minimum Clique Weight Heuristic method for graph triangulation. Thanks to Steffen Lauritzen, Asger Roer Pedersen, Lars Relund Nielsen and Claus Dethlefsen for commenting on the manuscript and for making preliminary checks of `gRain`.

A Working with HUGIN net files

The `HUGIN` program (see `http://www.hugin.com`) is a commercial program for Bayesian networks. A limited version of `HUGIN` is freely available. With `HUGIN`, a BN can be saved in a specific format known as a `net` file (which is a text file). A BN saved in this format can be loaded into `R` using the `loadHuginNet` function and a BN in `R` can be saved in the `net` format with the `saveHuginNet` function.

`HUGIN` distinguishes between node names and node labels. Node names have to be unique; node labels need not be so. When creating a BN in `HUGIN` node names are generated automatically as `C1`, `C2` etc. The user can choose to give more informative labels or to give informative names. Typically one would do the former. Therefore `loadHuginNet` uses node labels (if given) from the netfile and otherwise node names.

This causes two types of problems. First, in `HUGIN` it is allowed to have e.g. spaces and special characters (e.g. "?") in variable labels. This is not permitted in `gRain`. If such a name is found by `loadHuginNet`, the name is converted as follows: Special characters are removed, the first letter after a space is capitalized and then spaces are removed. Hence the label "visit to Asia?" in a `net` file will be converted to "visitToAsia". Then same convention applies to states of the variables. Secondly, because node labels in the net file are used as node names in `gRain` we may end up with two nodes having the same name which is obviously not permitted. To resolve this issue `gRain` will in such cases force the node names in `gRain` to be the node names rather than the node labels from the net file. For example, if nodes `A` and `B` in a net file both have label `foo`, then the nodes in `gRain` will be denoted `A` and `B`. It is noted that in itself this approach is not entirely fool proof: If there is a node `C` with label `A`, then we have just moved the problem. Therefore the scheme above is applied recursively until all ambiguities are resolved.

B iNets and the LS algorithm

To make this paper self–contained, this section briefly outlines PNs and computations with PNs as given in LS. Readers familiar with the algorithm can safely skip this section. The outline is based on the chest clinic example of LS which is illustrated in Figure 1.

B.1 Propagation

The LS algorithm allows conditional distributions to be calculated in a very efficient way, i.e. without first calculating the joint distribution and then carry out the marginalizations. Efficient propagation in `iNets` is based on representing the joint distribution (1) in different forms. These forms are derived from modifying the DAG. We describe these steps in the following but refer to Lauritzen and Spiegelhalter (1988) for further details as well as for references.

B.1.1 Compilation – from conditionals to clique potential presentation

The key to the computations is to transform the factorization in (2) into a clique potential representation: First the DAG is moralized which means that the parents of each node are joined by a line and then the directions on the arrows are dropped. Thus the moralized graph is undirected.

Next the moralized graph is triangulated if it is not already so. A graph is triangulated if it contains no cycles of length ≥ 4 without a chord. Triangulatedness can be checked using the Maximum Cardinality Search algorithm. If a graph is not triangulated it can be made so by adding edges, so called fill-ins. Finding an optimal triangulation of a given graph is NP–complete. Yet, various good heuristics exist. For graph triangulation we used the Minimum Clique Weight Heuristic method as described by Kjærulff (1990). Figure 2 shows the triangulated, moralized graph. We shall refer to the triangulated moralized DAG as the TUG.

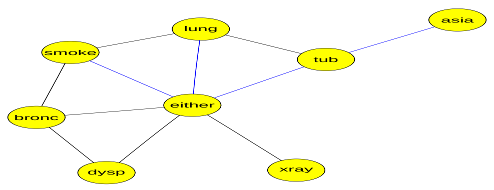

Figure 2: Triangulated moralized DAG – the chest clinic example from LS.

An ordering $C_1, \dots, C_T$ of the cliques of a graph has the Running Intersection Property (also called a RIP ordering) if $S_j = (C_1 \cup \dots C_{j-1}) \cap C_j$ is contained in one (but possibly several) of the cliques $C_1, \dots, C_{j-1}$. We pick one, say C_k and call this the parent clique of C_j while C_j is called a child of C_k. We call S_j the separator and $R_j = C_j \setminus S_j$ the residual, where $S_1 = \emptyset$. It can be shown that the cliques of a graph admit a RIP ordering if and only if the graph is triangulated.

The functions $p(v|pa(v))$ are hence defined on complete sets of the TUG. For each clique C we collect the conditional probability tables $p(v|pa(v))$ into a single term ψ_C by multiplying these conditional probability tables. Triangulation may have created cliques to which no CPT corresponds. For each such clique the corresponding potential is identical equal to 1. Thereby we obtain the *clique potential representation* of $p(V)$ as

$$p(V) = \prod_{j=1}^{T} \psi_{C_j}. \tag{3}$$

As such, a DAG and a corresponding factorization as in (2) is just one way of getting to the representation in (3).

B.1.2 Propagation – from clique potential to clique marginal representation

The propagation algorithm works by turning the clique potential representation into a clique marginal representation: To obtain the clique marginals $p(C_j)$ we proceed as follows. Start with the last clique C_T in the RIP ordering. The factorization (3) implies that $R_T \perp\!\!\!\perp (C_1 \cup \cdots \cup C_{T-1}) \setminus S_T | S_T$. Marginalizing over R_T gives

$$p(C_1 \cup \cdots \cup C_{T-1}) = [\prod_{j=1}^{T-1} \psi_{C_j}] \sum_{R_T} \psi_{C_T}.$$

Let $\psi_{S_T} = \sum_{R_T} \psi_{C_T}$. Then $p(R_T|S_T) = \psi_{C_T}/\psi_{S_T}$ and we have

$$P(V) = p(C_1 \cup \cdots \cup C_{T-1}) p(R_T|S_T) = \{[\prod_{j=1}^{T-1} \psi_{C_j}]\psi_{S_T}\}\psi_{C_T}/\psi_{S_T}.$$

Since ψ_{S_T} is a function defined on S_T and the RIP ordering ensures that S_T is contained in one of the cliques $C_1, \dots, C_{T-1}$, say C_k we can absorb ψ_{S_T} into ψ_{C_k} by setting $\psi_{C_k} \leftarrow \psi_{C_k}\psi_{S_T}$. After this absorption we have $p(C_1 \cup \dots C_{T-1}) = \prod_{j=1}^{T-1} \psi_{C_j}$. We can then apply the same scheme to this distribution to obtain $p(R_{T-1}|S_{T-1})$. Continuing this way backward gives

$$p(V) = p(C_1)p(R_2|S_2)p(R_3|S_3)\dots p(R_T|S_T) \tag{4}$$

where $p(C_1) = \psi_{C_1}/\sum_{C_1} \psi_{C_1}$. This is called a *set chain representation.*

Now we work forward. Suppose C_1 is the parent of C_2. Then $p(S_2) = \sum_{C_1 \setminus S_2} p(C_1)$ and so $p(V) = p(C_1)p(C_2)p(R_3|S_3)\dots p(R_T|S_T)/p(S_2)$. Proceeding this way yields the *clique marginal representation*

$$p(V) = \prod_{j=1}^{T} p(C_j) / \prod_{j=2}^{T} p(S_j). \tag{5}$$

Based on this representation, marginal probabilities of each node can be found by summing out over the other variables.

B.2 Absorbing evidence

Consider entering evidence $E = e^*$. We note that $P(V \setminus E | E = e^*) \propto p(V \setminus E, E = e^*)$. Hence evidence can be absorbed into the model by modifying the terms ψ_{C_j} in the clique potential representation (3): Entries in ψ_{C_j} which are inconsistent with the evidence $E = e^*$ are set to zero. We then proceed by carrying out the propagation steps above leading to (5) where the terms in the numerator then becomes $p(C_j|E = e^*)$. In this process we note that $\sum_{C_1} \psi_{C_1}$ is $p(E = e^*)$. Hence the probability of the evidence comes at no extra computational cost

B.3 Answering queries to BNs

To obtain $p(v|E = e^*)$ for some $v \in V \setminus E$, we locate a clique C_j containing v and marginalize as $\sum_{C_j \setminus \{v\}} p(C_j)$. Suppose we want the distribution $p(U|E = e^*)$ for a set $U \subset V \setminus E$. If there is a clique C_j such that $U \subset C_j$ then the distribution is simple to find by summing $p(C_j)$ over the variables in $C_j \setminus U$. If no such clique exists we can obtain $p(U|E = e^*)$ by calculating $p(U = u^*, E = e^*)$ for all possible configurations u^* of U and then normalize the result which is computationally demanding if U has a large state space. However, if it is known on beforehand that interest often will be in the joint distribution of a specific set U of variables, then one can ensure that the set U is in one clique in the TUG. The potential price to pay is that the cliques can become very large.

References

Dethlefsen, C. and Højsgaard, S. (2005). A common platform for graphical models in R: The gRbase package. *Journal of Statistical Software*, **14**, 1–12.

Green, P. J. (2005). GRAPPA: R functions for probability propagation.

Kjærulff, U. (1990). Triangulation of Graphs – Algorithms Giving Small Total State Space. Technical Report R 90-09, Aalborg University, Department of Mathematics and Computer Science, Fredrik Bajers Vej 7, DK 9220 Aalborg Ø, Denmark.

Lauritzen, S. L. and Spiegelhalter, D. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, **50**, (2), 157–224.

R Development Core Team (2007). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3.