Journal of Statistical Software

March 2012, Volume 46, Issue 10. *http://www.jstatsoft.org/*

Graphical Independence Networks with the gRain Package for R

Søren Højsgaard
Aalborg University

Abstract

In this paper we present the R package **gRain** for propagation in graphical independence networks (for which Bayesian networks is a special instance). The paper includes a description of the theory behind the computations. The main part of the paper is an illustration of how to use the package. The paper also illustrates how to turn a graphical model and data into an independence network.

Keywords: conditional independence, conditional probability, directed acyclical graph, evidence, expert system, graph, graphical model, junction tree, maximum cardinality search, message passing, running intersection property, triangulation.

1. Introduction

The **gRain** package (Højsgaard 2012a) is an R package, (R Development Core Team 2011) for probability propagation in *gRa*phical *i*ndependence *n*etworks which are also denoted *probabilistic networks* or *Bayesian networks*. **gRain** is available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=gRain`.

A Bayesian network is often taken to be a graphical model based on a directed acyclic graph (DAG). The DAG, however, is only used to give a simple way of specifying a multivariate distribution by combining (conditional) univariate distributions. The key to efficient probability propagation is to exploit conditional independencies in an undirected graph which is derived from the DAG. Hence, methods for building undirected graphical models can also be used for constructing probabilistic networks. Put differently, if we infer an undirected graphical model from data then we can convert the model to a probabilistic network and use this network for computation of conditional probabilities. As a concrete example, Tamayo *et al.* (2011) used **gRain** for probability propagation in model for clinical prediction in brain cancer. Throughout

the paper we shall use the term *network* for a graphical independence network.

The networks available in **gRain** are restricted to consisting of discrete variables, each with a finite state space. The propagation algorithm currently available in **gRain** is that of Lauritzen and Spiegelhalter (1988) which we denote the LS algorithm. For brevity we refer to the paper Lauritzen and Spiegelhalter (1988) as L&S.

To our knowledge there are two other R packages for probability propagation. One is the **GRAPPA** suite of functions (Green 2009) and the other is **RHugin** (Konis 2011). Neither of these packages are on CRAN.

The paper is organized as follows: Section 2 briefly reviews Bayesian networks through the chest clinic example of L&Sand the section also provides a very short presentation of some of the functionality of **gRain**. Section 3 describes the concepts and computational steps of L&Sand Section 4 then describes the **gRain** implementation of these steps. Section 5 describes methods for building networks from data and Section 6 describes additional features of **gRain** which includes methods for prediction, simulation and building networks from repeated patterns. Finally, Section 7 gives an overview of the software structure while Section 8 contains a discussion.

2. An introductory example: The chest clinic

This section reviews the chest clinic example of L&S (illustrated in Figure 1) and shows one way of specifying the network in **gRain**. Details of the steps will be given in Section 4 (additional ways of specifying a network are described in Section 5). L&S motivate the chest clinic example by 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."

2.1. Specifying a network through conditional probability tables

Let $X = X_V = (X_v; v \in V)$ be a discrete random vector where V is a set of labels of the variables. The levels of X_v are generically denoted by x_V so the levels of X are denoted $x = x_V = (x_v, v \in V)$. Each variable X_v has a finite number of levels.

In this section we outline one way of building a network. The starting point is a multivariate probability distribution constructed by combining univariate (conditional) distributions using the structure of a directed acyclic graph (hereafter denoted a DAG) with vertices V. The parents of a vertex v is denoted $pa(v)$. A joint distribution can be given as

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

where $p(x_v|x_{pa(v)})$ is a function defined on $X_v \times X_{pa(v)}$. This function is non-negative and satisfies that $\sum_{x_v} p(x_v|x^*_{pa(v)}) = 1$ for each parent configuration $x^*_{pa(v)}$ of $x_{pa(v)}$. Hence $p(x_v|x_{pa(v)})$

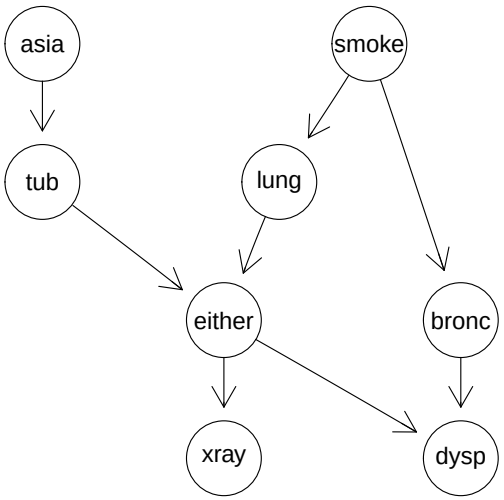

Figure 1: Chest clinic example from L&S.

becomes the conditional distribution of X_v given $X_{pa(v)}$. In practice $p(x_v|x_{pa(v)})$ is specified as a table called a conditional probability table (hereafter denoted a CPT). Thus, a Bayesian network can be regarded as a complex stochastic model built up by putting together simple components (conditional probability distributions).

For the chest clinic example we write the variables as $A = Asia$, $S = smoker$, $T = tuberculosis$, $L = lung\ cancer$, $B = bronchitis$, $D = dyspnoea$, $X = X\text{-}ray$ and $E = either\ tuberculosis\ or\ lung\ cancer$. Notice that we are using X with two different meanings here. Notice also that E is a logical variable which is true if either T or L are true and false otherwise.

Following (1), the DAG in Figure 1 dictates a factorization of the joint probability function $p(x_V)$ as

$$p(x_V) = p(x_A)p(x_T|x_A)p(x_S)p(x_B|x_S)x(x_L|x_S)p(x_E|x_T, x_L)p(x_D|x_E, x_B)p(x_X|x_E). \quad (2)$$

2.2. Queries to networks

Suppose we are given a finding that a set of variables $E \subset V$ have a specific value x_E^*. With this finding we are often interested in the conditional distribution $p(x_v|X_E = x_E^*)$ (or $p(x_v|x_E^*)$ in short) for some of the variables $v \in V \setminus E$ or in $p(x_U|x_E^*)$ for a set of variables $U \subset V \setminus E$. Interest might also be in calculating the probability of a specific event, e.g., the probability of seeing a specific finding, i.e., $p(x_E^*) = p(X_E = x_E^*)$.

In the chest clinic example, a finding x_E^* could be a person who has recently visited Asia and suffers from dyspnoea, i.e., $x_A =$ yes and $x_D =$ yes. Notice that E is used with two different meanings in this example. Interest might be in $p(x_L|x_E^*)$, $p(x_T|x_E^*)$ and $p(x_B|x_E^*)$, or possibly in the joint (conditional) distribution $p(x_L, x_T, x_B|x_E^*)$.

2.3. A one-minute version of gRain

A simple way of specifying the network for the chest clinic example is as follows (the steps presented below will be explained in detail in Section 4).

```
R> library("gRain")
R> library("Rgraphviz")
```

1. Specify conditional probability tables with values as given in L&S:

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

   Notice that the + operator used above is slightly misleading in the sense, for example, that the operator does not commute (the order of the variables *is* important). We use the + operator merely as a separator of the variables. The following forms are also valid specifications:

```
R> cptable(~ dysp | bronc + either,
+    values = c(9, 1, 7, 3, 8, 2, 1, 9), levels = yn)
R> cptable(c("dysp", "bronc", "either"),
+    values = c(9, 1, 7, 3, 8, 2, 1, 9), levels = yn)
```

   Notice also that since E is a logical variable which is true if either T or L are true and false otherwise, the corresponding CPT can be created with the special function `ortable()`.

2. Create an intermediate representation of the CPTs:

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

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 )
```

   For example we have

```
R> plist$tub
```

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

3. Create the network from the list of CPTs:

```
R> gin1 <- grain(plist)
R> summary(gin1)

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

4. The network can be queried:

```
R> querygrain(gin1, 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.

```
R> querygrain(gin1, nodes = c("lung", "bronc"), type = "joint")

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

5. Findings can be entered:

```
R> gin1.find <- setFinding(gin1, nodes = c("asia", "dysp"),
+    states = c("yes", "yes"))
```

6. The network with the findings incorporated can be queried:

```
R> querygrain(gin1.find, nodes = c("lung", "bronc"))

$lung
lung
       yes         no
0.09952515 0.90047485
```

```
$bronc
bronc
      yes        no
0.8114021 0.1885979
```

7. The probability of the specific finding can be obtained:

```
R> getFinding(gin1.find)

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

3. Local computations

This section describes the algorithm for making efficient computations with networks while the sections to follow will relate the facilities in **gRain** to the theory.

Returning to the chest clinic example, suppose interest is in $p(x_L|x_D = \text{“yes”}) = p(x_L, x_D = \text{“yes”})/p(x_D = \text{“yes”})$, i.e., the probability of lung cancer given dyspnoea. A brute force approach is to (i) carry out the multiplications in (2) (which would yield a 2^8 dimensional table) and then (ii) marginalize the table onto the 2^2 table defined by L and D. Such an approach is intractable for larger networks.

The key to making efficient computations with networks is (a) to utilize the modular structure of the model so that computations can be made locally and (b) to use the graph for identifying this modular structure.

3.1. Conditional independence and graphs

Before describing the local computations in detail, we review the notions of conditional independence and dependency graphs; see Lauritzen (1996) for further details. Let $X = (X_v)_{v\in V}$ be a random vector with a joint density. Let A, B and C be subsets of V and let $X_A = (X_v)_{v\in A}$ and similarly for X_B and X_C. Then X_A and X_B are conditionally independent given X_C (written $A \perp\!\!\!\perp B \,|\, C$) if X_A and X_B are independent in the conditional distribution given $X_C = x_c$ for each possible value of x_C of X_C. If $p()$ denotes a generic density or probability mass function, $A \perp\!\!\!\perp B \,|\, C$ if $p(x_A, x_B \,|\, x_C) = p(x_A \,|\, x_C)p(x_B \,|\, x_C)$. An equivalent characterization is that the joint density of (X_A, X_B, X_C) factorizes as

$$p(x_A, x_B, x_C) = g(x_A, x_C)h(x_B, x_C), \tag{3}$$

that is, as a product of two non-negative functions $g()$ and $h()$, where $g()$ does not depend on x_B and $h()$ does not depend on x_A. This is known as the *factorization criterion*.

Let $\mathcal{G} = (V, E)$ be an undirected graph with cliques $C_1, \ldots C_k$. If $p(x_V)$ can be factorized as

$$p(x_V) = \prod_{i=1}^{T} \psi_{C_i}(x_{C_i})$$

for some functions $\psi_{C_1}() \ldots \psi_{C_T}()$ where $\psi_{C_j}()$ depends on x only through x_{C_j} then we say that $p()$ factorizes according to $\mathcal{G}$. If all the densities under a given model factorize according to $\mathcal{G}$, then $\mathcal{G}$ encodes the conditional independence structure of the model, through the following result (the *global Markov property*): whenever sets A and B are separated by a set C in $\mathcal{G}$, then $A \perp\!\!\!\perp B \,|\, C$ under the model.

3.2. The moral graph

We may think of each of the conditional probability tables of the form $p(x_v|x_{pa(v)})$ in (2) as a non-negative function of the variables it involves. We can make this explicit by writing $p(x_v|x_{pa(v)})$ as $\psi_{v\cup pa(v)}(x_v, x_{pa(v)})$ or even shorter as $\psi(x_v, x_{pa(v)})$. Following L&S we call these functions *evidence potentials* or simply *potentials* and we may hence write (2) as

$$p(x_V) = \psi(x_A)\psi(x_T, x_A)\psi(x_S)\psi(x_B, x_S)\psi(x_L, x_S)\psi(x_E, x_T, x_L)\psi(x_D, x_E, x_B)\psi(x_X, x_E),$$

where $\psi(x_A) = p(x_A)$, $\psi(x_T, x_A) = p(x_T|x_A)$, $\psi(x_S) = p(x_S)$, $\psi(x_B, x_S) = p(x_B|x_S)$ and so forth. Absorbing lower order terms into higher order terms allows us to write (2) as

$$p(x_V) = \psi(x_A, x_T)\psi(x_S, x_B)\psi(x_S, x_L)\psi(x_E, x_T, x_L)\psi(x_D, x_E, x_B)\psi(x_X, x_E) \tag{4}$$

where, for example, $\psi(x_A, x_T) = p(x_A)p(x_T|x_A)$.

The dependence graph of a Bayesian network is an undirected graph with vertices V, and this graph is derived from these potentials. For example, the presence of the term $p(x_D|x_E, x_B)$ implies that there must be edges between all pairs in $\{D, E, B\}$. The dependence graph can algorithmically be formed from the DAG by moralization: The moral graph of a DAG is obtained by first joining all parents of each node by an edge and then dropping directions on the arrows. The moral graph for the chest clinic example is shown in Figure 2, left. The edges added are between `tub` and `lung` and between `either` and `bronc`. The global Markov property allows conditional independence restrictions implied by (4) to be read off the moral graph.

Next we illustrate an approach to efficient computations. To find e.g., $p(x_L, x_D)$ we need to sum over all other variables. We are interested in making these summations locally so that we do not create new potentials defined over large sets of variables because this is prohibitive for large networks.

For example if we start by summing over x_T then it follows from (4) that we will create a potential which depends on (x_A, x_L, x_E), namely $\psi(x_A, x_L, x_E) = \sum_{x_T} \psi(x_T, x_A)\psi(x_E, x_T, x_L)$. Summing over x_A afterwards then amounts to finding $\sum_{x_A} \psi(x_A, x_L, x_E)$.

On the other hand, if we start by summing over x_A then this means summing in a single potential, namely $\psi^*(x_T) = \sum_{x_A} \psi(x_T, x_A)$. This new potential can then be *absorbed* into $\psi(x_E, x_T, x_L)$ by setting $\psi(x_E, x_T, x_L) \leftarrow \psi(x_E, x_T, x_L)\psi^*(x_T)$. Thereby no new potential has been created; only an existing potential has been modified. Summing over x_T afterwards then amounts to finding $\psi^*(x_E, x_L) = \sum_{x_T} \psi^*(x_E, x_T, x_L)$ which again is a sum involving only a single potential. We can also employ such local computations when summing over x_X and it then follows that

$$p(x_L, x_E, x_S, x_B, x_D) = \psi(x_L, x_S)\psi(x_B, x_S)\psi(x_E, x_L)\psi(x_D, x_E, x_B).$$

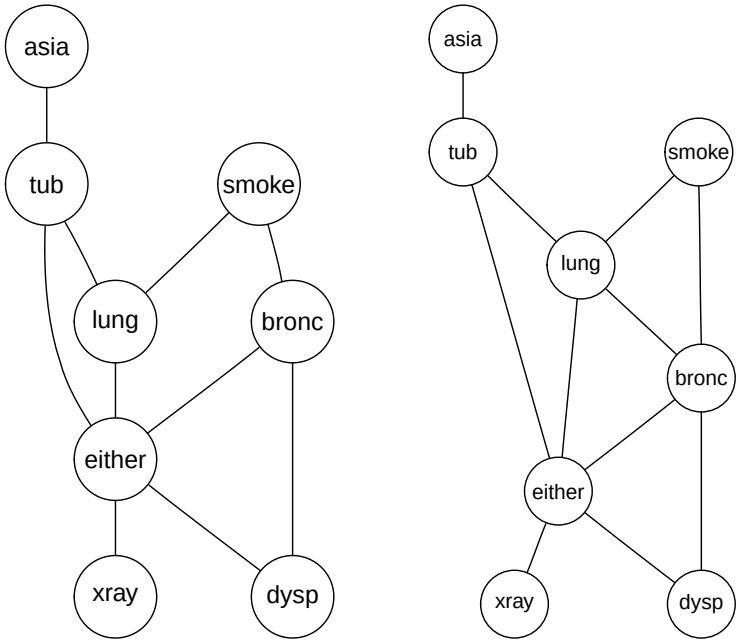

Figure 2: Left: Moral graph obtained by adding edges between common parents of each node and then dropping directions. Right: Triangulated moralized DAG.

Next we will have to sum over either x_S, x_B or x_E. Summing over x_S gives

$$p(x_L, x_E, x_B, x_D) = \psi(x_E, x_L)\psi(x_D, x_E, x_B) \sum_{x_S} \psi(x_L, x_S)\psi(x_B, x_S). \tag{5}$$

This will produce a function of (x_L, x_B), namely $\psi(x_L, x_B) = \sum_{x_S} \psi(x_L, x_S)\psi(x_B, x_S)$. Since L and B are not neighbours in the moral graph there is no potential into which $\psi(x_L, x_B)$ can be absorbed. (Summing over x_B or x_E would also create functions of variables which are not neighbours in the moral graph).

3.3. Triangulation

There is a simple condition under which the summations (when carried out in a suitable order) can be made so that no new functions are created. The condition is that the moral graph is triangulated. A graph is triangulated if and only if the graph contains no cycles of length ≥ 4 without a chord. If a graph is not triangulated it can be made so by adding edges, so called fill-ins. A triangulation of the moral graph for the chest clinic example is shown in Figure 2, right (an edge between `bronc` and `lung` has been added). Moreover, when the graph is triangulated it is possible to give an ordering in which all summations can be made such that no new functions are created. We shall return to describing the ordering later.

Clearly, $p(x_V)$ has interactions only among neighbours in the triangulated graph and we may write

$$p(V) = \psi(x_T, x_A)\psi(x_L, x_B, x_S)\psi(x_L, x_T, x_S)\psi(x_E, x_B, x_L)\psi(x_D, x_E, x_B)\psi(x_X, x_E). \tag{6}$$

where $\psi(x_L, x_B, x_S) = p(x_S)p(x_L|x_S)p(x_B|x_S)$, $\psi(x_E, x_B, x_L) = 1$ and all other potentials are as defined in connection with (4). The representation in (6) is called a *clique potential representation*. Using (6), the analogue of (5) becomes to set $\psi(x_E, x_B, x_L) \leftarrow \psi(x_E, x_B, x_L) \sum_{x_S} \psi(x_L, x_B, x_S)$ so in this case there is now an existing potential $\psi(x_E, x_B, x_L)$ into which $\sum_{x_S} \psi(x_L, x_B, x_S)$ can be incorporated. We then obtain

$$p(x_L, x_E, x_B, x_D) = \psi(x_E, x_L)\psi(x_D, x_E, x_B)\psi(x_E, x_B, x_L). \tag{7}$$

3.4. Different representations of the joint distribution

The key to efficient calculations is to transform the clique potential representation (such as the representation in (6)) into a clique marginal representation which is described below.

To do so, the following graph theoretical concepts are needed (where we refer to L&S for additional details and for references): Let $bd(v_i)$ denote the neighbours of the vertex v_i in an undirected graph. A set of variables U in an undirected graph is complete if there are edges between all pairs of variables in U. A numbering $v_1, \ldots, v_k$ of the vertices is perfect if and only if for all i the sets $\{v_1, \ldots, v_{i-1}\} \cap bd(v_i)$ are complete. It can be shown that a graph is triangulated if the vertices can be given a perfect ordering. Triangulatedness can be checked using the maximum cardinality search (abbreviated MCS) algorithm which works as follows: Give number 1 to any node and proceed iteratively as follows: The next node to number is a node with a maximum of previously numbered neighbours. If the graph is triangulated then the numbering obtained this way will be perfect. If the graph is not triangulated it can be made so by adding additional 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 have in **gRain** used the minimum clique weight heuristic method suggested by Kjærulff (1990).

An ordering $C_1, \ldots, C_T$ of the cliques of an undirected graph has the Running Intersection Property if $S_j = (C_1 \cup \ldots C_{j-1}) \cap C_j$ is contained in one (but possibly several) of the cliques $C_1, \ldots, C_{j-1}$. An ordering of the cliques that satisfies the Running Intersection Property is also called a RIP ordering. We pick one of the cliques, say C_k of $C_1, \ldots C_{j-1}$ 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. A RIP ordering can be found, for example, by ordering the cliques according to the highest number assigned to the vertices in each clique by the maximum cardinality search algorithm.

Table 1 shows the RIP ordering of the cliques of the triangulated moral graph for the chest clinic example as obtained in **gRain**.

The RIP ordering of the cliques can be represented as the DAG in Figure 3 which we call a rooted junction tree with C_1 as the root.

In a general setting we may (following the steps illustrated above) obtain a *clique potential representation* of $p(x_V)$ given in (1) as

$$p(x_V) = \prod_{j=1}^{T} \psi_{C_j}(x_{C_j}) \tag{8}$$

where $C_1, C_2, \ldots, C_T$ are the cliques of the triangulated moral graph.

Clique no.	Clique	Separator	Parent
1	tub, asia		
2	either, tub, lung	tub	1
3	bronc, lung, either	lung, either	2
4	smoke, lung, bronc	lung, bronc	3
5	dysp, bronc, either	bronc, either	3
6	xray, either	either	5

Table 1: RIP ordering of the cliques of the triangulated moral graph for the chest clinique example. For example, clique number 4 has clique number 3 as its parent.

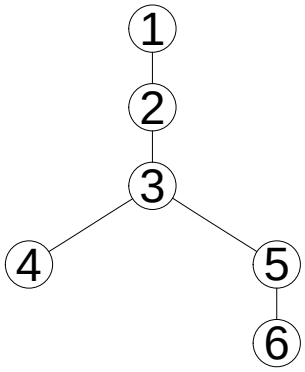

Figure 3: Rooted junction tree for the chest clinic example from L&S. The numbers refer to the cliques given in Table 1.

Notice that (1) can be seen as a way of specifying a complex joint distribution through simpler terms, a collection of CPTs. However, these CPTs are only used for establishing the clique potential representation (8). Once this representation is established, the CPTs are of no further use. It should also be noticed that the clique potentials are not uniquely defined (and in the following we shall make use of alternative sets of clique potentials).

The LS algorithm works by turning the clique potential representation (8) into a representation in which each potential $\psi_{C_j}(x_{C_j})$ is replaced by the marginal distribution $p(x_{C_j})$. This representation is called a clique marginal representation. This is done by working twice through the set of cliques and passing messages between neighbouring cliques: first from the last clique in the RIP ordering towards the first, i.e., inwards in the junction tree, and subsequently passing messages in the other direction.

In detail, we proceed as follows. Start with the last clique C_T in the RIP ordering where $C_T = S_T \cup R_T$, $S_T \cap R_T = \emptyset$ (for the chest clinic example, this is C_6 in Figure 3). The factorization (8) together with the factorization criterion (3) implies that $R_T \perp\!\!\!\perp (C_1 \cup \ldots \cup C_{T-1}) \setminus S_T \,|\, S_T$. Marginalizing over x_{R_T} gives

$$p(x_{C_1 \cup \ldots \cup C_{T-1}}) = \Big(\prod_{j=1}^{T-1} \psi_{C_j}(x_{C_j}) \Big) \sum_{x_{R_T}} \psi_{C_T}(x_{S_T}, x_{R_T}).$$

Let $\psi_{S_T}(x_{S_T}) = \sum_{x_{R_T}} \psi_{C_T}(x_{S_T}, x_{R_T})$. Then from the expression above we have

$$p(x_{R_T} \,|\, x_{S_T}) = \psi_{C_T}(x_{S_T}, x_{R_T}) / \psi_{S_T}(x_{S_T})$$

and hence

$$p(x_V) = p(x_{C_1 \cup \ldots \cup C_{T-1}}) p(x_{R_T} \,|\, x_{S_T}) = \left\{ \left(\prod_{j=1}^{T-1} \psi_{C_j}(x_{C_j}) \right) \psi_{S_T}(x_{S_T}) \right\} \frac{\psi_{C_T}(x_{C_T})}{\psi_{S_T}(x_{S_T})}.$$

The RIP ordering ensures that S_T is contained in the neighbour of C_T in the junction tree (one of the cliques $C_1, \ldots, C_{T-1}$), say C_k. (In Figure 3, S_6 is contained in C_5.) We can therefore absorb ψ_{S_T} into ψ_{C_k} by setting $\psi_{C_k}(x_{C_k}) \leftarrow \psi_{C_k}(x_{C_k}) \psi_{S_T}(x_{S_T})$. We can think of the clique C_T passing the message ψ_{S_T} to its neighbour C_k, and then changing its own potential to $\psi_{C_T} \leftarrow \psi_{C_T} / \psi_{S_T}$. Similarly, C_k absorbes the message. After this we now have $p(x_{C_1 \cup \ldots \cup C_{T-1}}) = \prod_{j=1}^{T-1} \psi_{C_j}(x_{C_j})$. We can then apply the same scheme to the part of the junction tree which has not yet been traversed. Continuing in this way until we reach the root of the junction tree yields

$$p(x_V) = p(x_{C_1}) p(x_{R_2} \,|\, x_{S_2}) p(x_{R_3} \,|\, x_{S_3}) \ldots p(x_{R_T} \,|\, x_{S_T}) \tag{9}$$

where $p(x_{C_1}) = \psi_{C_1}(x_{C_1}) / \sum_{x_{C_1}} \psi_{C_1}(x_{C_1})$. The resulting expression (9) is called a *set chain representation*. Note that the root potential now yields the joint marginal distribution of its nodes. (In Section 5 we establish the representation (9) directly from a triangulated graph and a dataset).

Next we the other way in the set chain representation (and outwards in the rooted junction tree): Suppose C_1 is the parent of C_2 in the rooted junction tree. Then we have that $p(x_{S_2}) = \sum_{x_{C_1 \setminus S_2}} p(x_{C_1})$ and so

$$p(x_V) = p(x_{C_1}) \frac{p(x_{C_2})}{p(x_{S_2})} p(x_{R_3} \,|\, x_{S_3}) \ldots p(x_{R_T} \,|\, x_{S_T}).$$

Thus when the clique C_2 has absorbed its message by the operation

$$\psi_{C_2}(x_{C_2}) \leftarrow \psi_{C_2}(x_{C_2}) p(x_{S_2})$$

its potential is equal to the marginal distribution of its nodes. Proceeding this way until we reach the leaves of the junction tree yields the *clique marginal representation*

$$p(x_V) = \prod_{j=1}^{T} p(x_{C_j}) / \prod_{j=2}^{T} p(x_{S_j}). \tag{10}$$

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

3.5. Absorbing evidence and answering queries

Consider absorbing the evidence $x_E^* = (x_v^*, v \in E)$, i.e., that nodes in E are known to have a specific value. We note that

$$p(x_{V \setminus E} \,|\, x_E^*) \propto p(x_{V \setminus E}, x_E^*)$$

and hence evidence can be absorbed into the model by modifying the terms ψ_{C_j} in the clique potential representation (8) as follows: for every $v \in E$, we pick an arbitrary clique C_j with $v \in C_j$. Entries in ψ_{C_j} which are locally inconsistent with the evidence, i.e., entries x_{C_j} for which $x_v \neq x_v^*$, are set to zero and all other entries are unchanged. Evidence can be entered before or after propagation without changing final results.

To answer queries, we carry out the propagation steps above leading to a clique marginal representation where the potentials become $\psi_{C_j}(x_{C_j}) = p(x_{C_j}|x_E^*)$. In this process we note that processing of the root potential to find $p(x_{C_1} \,|\, x_E^*)$ involves computation of $\sum_{x_{C_1}} \psi_1(x_{C_1})$ which is equal to $p(x_E^*)$. Hence the probability of the evidence comes at no extra computational cost.

Suppose we want the distribution $p(x_U \,|\, x_E^*)$ 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 $p(x_{C_j})$ over the variables in $C_j \setminus U$. If no such clique exists we can obtain $p(x_U \,|\, x_E^*)$ by calculating $p(x_U^*, x_E^*)$ for all possible configurations x_U^* of U and then normalize the result: this can be computationally demanding if U has a large state space.

4. Local computations in gRain

This section illustrates how the local computation steps described in Section 3 are made in **gRain**. The two main concepts are *compilation* and *propagation*; a terminology which has been adopted from the **HUGIN** software (**HUGIN** Expert A/S 2011).

In general terms, *compilation* refers to a transformation of specification of a joint probability distribution into a *clique potential representation* of the form (8) while *propagation* means transforming a clique potential representation of the form (8) into the clique marginal representation (10).

4.1. Specification of a network

Consider the following code fragment (which is also used in Section 2.3):

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


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 )
```

The `compileCPT()` function performs the following steps: (i) Checks that specifications of the CPTs do not specify cycles. (ii) Create the corresponding DAG. (iii) Checks that the dimensions of the CPTs are consistent. (For example the specification of `t.a` gives a table

with four entries and the variable `tub` is specified to be binary. Hence it is checked that the variable `asia` is also binary).

A network is specified using the generic `grain()` function. There is a `grain()` method for different types of model specification. The `grain()` method applied to `plist` (as defined above) sets up an internal structure which is subsequently used in the computations:

```
R> gin1  <- grain(plist)
R> summary(gin1)

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

4.2. Compilation and propagation

Compilation

The `compile()` method performs the following steps: (i) Creates the moral graph. (ii) Detects that the moral graph is not triangulated and therefore creates a triangulated graph by making fill-ins. (iii) Establishes a potential representation by absorbing each CPT into an appropriate clique potential; i.e., establish the representation in (8. (iv) Creates a junction tree of the cliques.

```
R> gin1c <- compile(gin1)
R> summary(gin1c)

Independence network: Compiled: TRUE Propagated: FALSE
 Nodes : chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
 Number of cliques:                 6
 Maximal clique size:               3
 Maximal state space in cliques:    8
```

A RIP ordering of the cliques of the triangulated undirected graph is contained in the slot `rip` in the network object; see Table 1.

Propagation

Propagation means transforming a clique potential representation of the form (8) into the clique marginal representation (10):

```
R> gin1cp <- propagate(gin1c)
```

Notice that the `compile()` and `propagate()` functions were not called explicitly in the example in Section 2.3. The reason for this is given in Section 4.3.

4.3. Setting findings and answering queries

Setting findings

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

```
R> gin1c.find <- setFinding(gin1c, nodes = c("asia", "dysp"),
+    states = c("yes", "yes"))
R> gin1c.find <- setFinding(gin1c,
+    flist = list(c("asia", "yes"), c("dysp", "yes")))
```

These findings are entered into the clique potential representation (8) as described in Section 3.5. If the network is not already compiled then no clique potential representation exists for the network. In this case, `setFinding()` will force a compilation to be made. Default is that the network is also propagated when `setFinding()` is called and therefore the `compile()` and `propagate()` functions were not called in Section 2.3. Notice that findings can be entered incrementally by calling `setFinding()` repeatedly. If doing so, it is advantageous (in terms of computing time) to set `propagate = FALSE` in `setFinding()` and then only call the `propagate()` function at the end:

```
R> gin1c.find <- setFinding(gin1c, nodes = "asia", states = "yes",
+    propagate = FALSE)
R> gin1c.find <- setFinding(gin1c.find, nodes = "dysp", states = "yes",
+    propagate = FALSE)
R> gin1c.find <- propagate(gin1c.find)
```

The finding itself is displayed with:

```
R> getFinding(gin1c.find)
```

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

The probability of observing the finding is obtained with `pFinding()`. Findings can be retracted (removed from the network) with

```
R> gin1c3 <- retractFinding(gin1c.find, nodes = "asia")
R> getFinding(gin1c3)
```

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

Omitting the `nodes` argument when calling `retractFinding()` implies that all findings are retracted, i.e., that the network is reset to its original status.

Answering queries

As illustrated in Section 2.3, queries can be made to a network using the `querygrain()` function. The result is by default an array (or a list of array(s)). Setting `result = "data.frame"` causes the result to be returned as a dataframe (or a list of dataframes). Calling `querygrain()` on a network which is not propagated will force a propagation to be made.

For example, consider the network with the findings inserted above:

```
R> querygrain(gin1c.find, nodes = c("lung", "bronc"), type = "marginal",
+    result = "data.frame")

$lung
  lung       Freq
1  yes 0.09952515
2   no 0.90047485

$bronc
  bronc      Freq
1   yes 0.8114021
2    no 0.1885979

R> querygrain(gin1c.find, nodes = c("lung", "bronc"), type = "joint",
+    result = "data.frame")

  lung bronc       Freq
1  yes   yes 0.06298076
2   no   yes 0.74842132
3  yes    no 0.03654439
4   no    no 0.15205354

R> querygrain(gin1c.find, nodes = c("lung", "bronc"), type = "conditional",
+    result = "data.frame")

  lung bronc       Freq
1  yes   yes 0.07761966
2   no   yes 0.92238034
3  yes    no 0.19376877
4   no    no 0.80623123
```

In the first instance where `type = "marginal"` we get $p(x_L|\text{finding})$ and $p(x_B|\text{finding})$; notice that `type = "marginal"` is the default. Next, setting `type = "joint"` gives $p(x_L, x_B|\text{finding})$. Finally, setting `type = "conditional"` gives $p(x_L|x_B, \text{finding})$, i.e., the distribution of the first variable in `nodes` given the remaining variables listed.

Omitting `nodes` when calling `querygrain()` implies that all nodes are considered. Hence omitting `nodes` and setting `type = "joint"` means that the full table will be calculated.

5. Building networks from data

The CPTs used in the previous sections have all been specified directly. However, it is clearly possible to extract such tables from data. Below we illustrate two approaches for doing so.

5.1. From a directed acyclic graph

This section describes building a network from a given DAG and data. As an example we consider the coronary artery disease data in the dataframe `cad1` (see Højsgaard and Thiesson 1995 for a more detailled discussion of the data). Put briefly the coronary artery disease `CAD (C)` is the response variable. Occurrence of `CAD` is believed to depend on smoking habits (`Smoker (S)`), hereditary predispositions (`Inherit (I)`) and hypercholesterolamia (`Hyperchol (H)`). Manifestations of the disease includes angina pectoris (`AngPec (A)`) and recordings of other heart failures (`Heartfail (F)`). Furthermore, the ECG reading of Q-wave (`QWave (Q)`) is a manifestation of `CAD`. We postulate the simple DAG structure in Figure 4:

```
R> cad.dag <- dag(~ CAD:Smoker:Inherit:Hyperchol + AngPec:CAD +
+    Heartfail:CAD + QWave:CAD)
```

Given a DAG and the data, it is possible to extract CPTs and create a network from these as follows:: The CPTs extracted are relative frequencies of each node given its parents. To

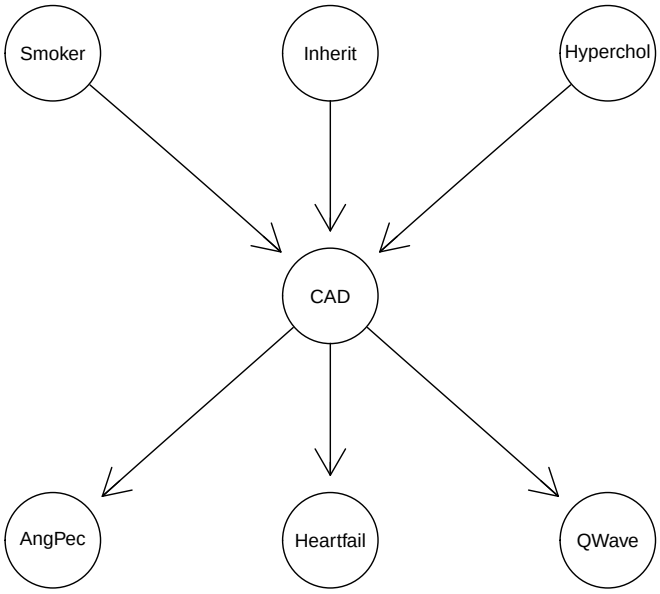

Figure 4: A DAG model for coronary artery disease data.

avoid zeros in the CPTs one can choose to add a small number, e.g., `smooth = 0.1` to all values, corresponding to a Bayesian estimate based on prior Dirichlet distributions for the CPT entries:

```
R> data("cad1")
R> cad.cpt <- extractCPT(cad1, cad.dag, smooth = 0.1)
R> cad.gin <- grain(compileCPT(cad.cpt))
```

These steps are wrapped into a single function which takes a DAG and a table as arguments:

```
R> grain(cad.dag, data = cad1, smooth = 0.1)

Independence network: Compiled: FALSE Propagated: FALSE
 Nodes: chr [1:7] "CAD" "Smoker" "Inherit" "Hyperchol" ...
```

5.2. From a triangulated undirected graph

In this section we describe how to build a network from an triangulated undirected graph. The situation we have in mind is where log-linear model for a contingency table has been found, for example using a stepwise model selection procedure.

Returning to the coronary artery disease data of Section 5.1 we can start with the saturated model for the seven variables and do a stepwise backward elimination among decomposable models. We do so using **gRim**, Højsgaard (2012b) where the model search is among decomposable models only and where the selection criterion is AIC:

```
R> library("gRim")
R> cad.tab <- xtabs(~ ., data = cad1)
R> cad.sat <- dmod(~ .^., data = cad.tab, marginal = c("CAD", "Smoker",
+    "Inherit", "Hyperchol", "AngPec", "Heartfail", "QWave"))
R> (cad.sel <- stepwise(cad.sat))

Model: A dModel with 7 variables
 graphical :  TRUE  decomposable :  TRUE
 -2logL    :        1972.91 mdim :   31 aic :      2034.91
 ideviance :         248.10 idf  :   23 bic :      2142.29
 deviance  :         140.30 df   :  160
Notice: Table is sparse
  Asymptotic chi2 distribution may be questionable.
  Degrees of freedom can not be trusted.
  Model dimension adjusted for sparsity : 30
```

The selected model is displayed with

```
R> plot(cad.sel, "circo")
```

and is shown in Figure 5. To turn this model into a network, two observations are made: First, the compilation step in Section 4.2 only served to get from a list of CPTs to the

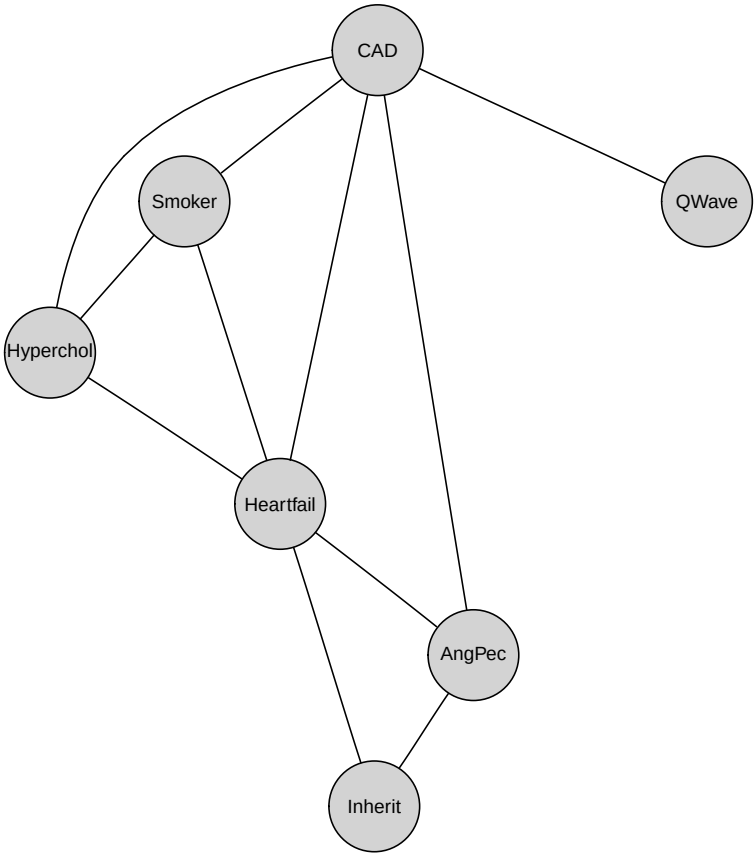

Figure 5: Decomposable log-linear model for coronary artery disease data.

clique potential representation (8) of the cliques of a triangulated undirected graph. After the clique potential representation is obtained, the CPTs are of no further use. Secondly, the set chain representation (9) is one such clique potential representation and obtaining this set chain representation is straight forward given data and a triangulated graph: Given the RIP ordering of the cliques we extract the tables for a clique C_k and a separator S_k, normalize these to sum to one to give $p(x_{C_k})$ and $p(x_{S_k})$ and set $\psi_{C_k}(x_{C_k}) = p(x_{C_k})/p(x_{S_k})$. These potentials are then turned into a network.

The model object `cad.sel` contains the generating class for the model and from this, a graph object can be constructed:

```
R> cad.gc <- cad.sel$glist
R> str(cad.gc)

List of 4
 $ : chr [1:4] "CAD" "Smoker" "Hyperchol" "Heartfail"
 $ : chr [1:3] "CAD" "AngPec" "Heartfail"
 $ : chr [1:3] "Inherit" "AngPec" "Heartfail"
 $ : chr [1:2] "CAD" "QWave"
```

```
R> (cad.ug <- ugList(cad.gc))

A graphNEL graph with undirected edges
Number of Nodes = 7
Number of Edges = 11
```

From the graph object `cad.ug` and data `cad1` we may now extract the clique potentials and turn these into a network:

```
R> potlist <- extractPOT(cad1, cad.ug)
R> grain(compilePOT(potlist))

Independence network: Compiled: FALSE Propagated: FALSE
 Nodes: chr [1:7] "Heartfail" "CAD" "Smoker" "Hyperchol" ...
```

These steps are wrapped into a single function which takes an undirected (triangulated) graph and data as arguments:

```
R> grain(cad.ug, data = cad1)
```

6. Additional features of gRain

6.1. Fast computation of a joint distribution

Suppose interest is in fast computation of a joint distribution of a set of variables U as discussed in Section 3.5. In this case one can force U to be in the same clique of the triangulated moralized DAG as:

```
R> gin1c.alt <- compile(gin1, root = c("lung", "bronc", "tub"),
+    propagate = TRUE)
R> summary(gin1c.alt)

Independence network: Compiled: TRUE Propagated: TRUE
 Nodes : chr [1:8] "asia" "tub" "smoke" "lung" "bronc" ...
 Number of cliques:                 5
 Maximal clique size:               4
 Maximal state space in cliques:   16
```

Hence this network has a larger clique size than the original network in Section 4.2. The triangulated graph resulting from this is shown in Figure 6.

The computing time for the joint distribution of `lung`, `bronc` and `tub` is much smaller when compared with the original network:

```
R> system.time({
+    for (i in 1:200)
+    querygrain(gin1c.alt, nodes = c("lung", "bronc", "tub"), type = "joint")
+  })
```

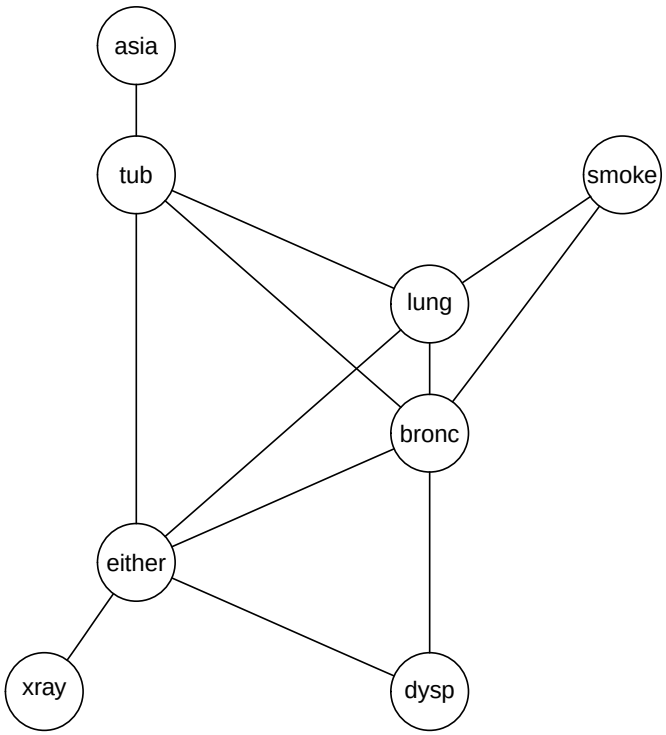

Figure 6: Triangulation when `lung`, `bronc` and `tub` are forced to be contained in one clique. This makes computation of their joint distribution faster but at the expense of a larger clique size.

```
   user  system elapsed
   0.03    0.00    0.03

R> system.time({
+    for (i in 1:200)
+    querygrain(gin1c, nodes = c("lung", "bronc", "tub"), type = "joint")
+  })

   user  system elapsed
   4.68    0.00    4.71
```

6.2. Simulation

It is possible to simulate data from a network (with or without findings) using the method of Dawid (1992), see also Cowell *et al.* (1999, p. 99). If a network is not propagated when `simulate()` is applied, `simulate()` will force this to happen automatically. The algorithm works as follows: Consider again Figure 3. First data are simulated for the variables in C_1 using $p(x_{C_1})$. Then data for $R_2 = C_2 \setminus S_2$ are generated from $p(x_{R_2}|x_{S_2})$ using the previously simulated data for $S_2 = C_2 \cap C_1$ and so on outwards in the junction tree.

```
R> sim.find <- simulate(gin1c.find, nsim = 5000)
R> head(sim.find)

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

Such a simulation may be useful, for example for finding a joint (conditional) distribution of a set of variables without having to take the approach illustrated in Section 6.1. For example we obtain the following approximation which is close to the exact result given in Section 4.3:

```
R> xtabs(~ lung + bronc, data = sim.find) / nrow(sim.find)

     bronc
lung     yes     no
  yes 0.0598 0.0374
  no  0.7524 0.1504
```

6.3. Prediction

A `predict` method is available for networks 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:

```
R> 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

R> predict(gin1c, 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"


$pFinding
[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 jointly most likely configuration of the two variables `lung` and `bronc` is `"yes"`.

Alternatively, one can obtain the entire (conditional) distribution:

```
R> predict(gin1c, 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


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

Above, `pFinding` is the probability of each configuration of the explanatory variables.

6.4. Repeated patterns

The **gRain** package provides a simple mechanism for producing repeated patterns. To illustrate this we define a homogeneous hidden Markov model

$$p(x, y) = p(x_0) \prod_{t=1}^{5} p(x_t|x_{t-1}) p(y_t|x_t)$$

where the x_ts are unobserved and the y_ts are observed.

First we define templates for transition and emission densities for each time point:

```
R> yn  <- c("yes", "no")
R> x.x <- cptable(~ x[i] | x[i-1], values = c(1, 99, 2, 98), levels = yn)
R> y.x <- cptable(~ y[i] | x[i], values = c(10, 90, 5, 95), levels = yn)
```

Next we create instances of these densities using `repeatPattern()`:

```
R> inst <- repeatPattern(list(x.x, y.x), instances = 1:5)
```

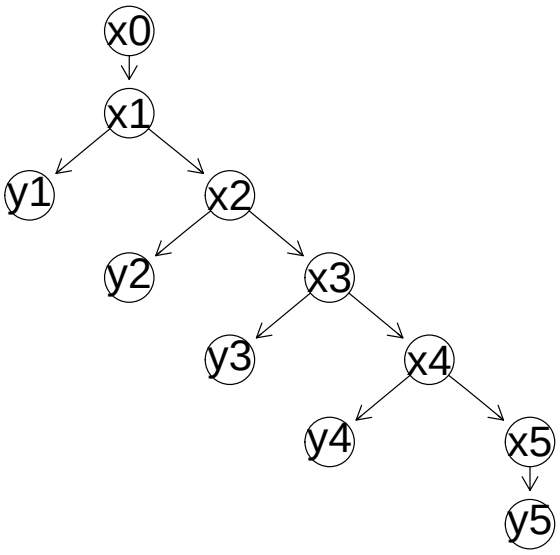

Figure 7: Hidden Markov model generated with repeated patterns.

Finally we combine these instances with the initial distribution $p(x_0)$ and create the network as shown in Figure 7:

```
R> x.0 <- cptable(~ x0, values = c(1, 1), levels = yn)
R> hmm <- compileCPT(c(list(x.0), inst))
R> hmm <- grain(hmm)
```

Højsgaard *et al.* (2012) shows an example where `repeatPattern()` is used in connection with parameter learning in a Bayesian setting.

6.5. Plotting networks

There are two sets of methods for plotting networks: (a) The `plot()` methods are based on the **Rgraphviz** package Gentry *et al.* (2012) which requires that the **Graphviz** program, (AT&T Research 2009) is installed. All plots in this paper are created using these `plot()` methods. (b) The `iplot()` methods are based on the **igraph** package (Csardi and Nepusz 2006) and **igraph** does not require external programs to be installed.

6.6. Working with HUGIN net files

The **HUGIN** program (**HUGIN** Expert A/S 2011) is a commercial program for Bayesian networks. A limited version of **HUGIN** is freely available. A network made in **HUGIN** can be saved in a specific format known as a `net` file (which is a text file). Such a `net` file can be read into R as a network using the `loadHuginNet()` function and a network 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 network 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 foolproof: 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.

7. Overview of software structure

In this section we briefly summarizes the software structure.

A network can be specified in different ways: (a) From a list of CPTs, (Section 2.3) (b) From a DAG and a dataset (Section 5.1) and (c) From a triangulated undirected graph and a dataset (Section 5.2). The `grain()` function is used in this connection.

Findings are entered into a network using `setFinding()` and queries to a network are made using `querygrain()` (see Sections 2.3 and 4.3). Findings can only be entered to a compiled network and queries can only be asked to a compiled and propagated network. However, compilation and propagation (using `compile()` and `propagate()`) is forced (when necessary) by the `setFinding()` and `querygrain()` functions).

Findings can be entered sequentially and findings can be retracted (Section 4.3). Entering findings sequentially is one example where it may be advantageous to defer propagation until all findings have been entered.

When compiling a network it is possible to force a set of variables to be in the same clique of the underlying undirected graph. Section 6.1 shows an example where this is used.

8. Discussion and perspectives

This paper describes a propagation algorithm for graphical independence networks (Bayesian networks); how to set findings and how to query such networks. The paper also describes how to establish networks from data and a statistical model.

The critical point in terms of storage is the size of the clique potentials which is determined by the size of the cliques in the underlying triangulated graph. The critical computational point in connection with computing time are the operations on clique potentials: multiplication, division and marginalization. When the cliques become large and/or the variables have many levels then **gRain** becomes slow. On the other hand, if the underlying graph is sparse (for example a tree) then **gRain** is quite fast. Thereby it is feasible to use **gRain** in

e.g., bioinformatic applications where the underlying graph often is sparse.

An alternative to **gRain** is the **RHugin** package, Konis (2011). **RHugin** provides an interface to the **HUGIN** Application Programmers Interface (**HUGIN** Expert A/S 2011) which makes **RHugin** very efficient. **RHugin** works with the limited version of **HUGIN** which is freely available. In terms of speed and computational efficiency, **RHugin** is much more efficient than **gRain**.

There exist faster algorithms than the one implemented in **gRain**, for example the algorithm described by Jensen *et al.* (1990). Their algorithm is presumably faster than the LS algorithm but also more expensive in terms of memory requirements than the LS algorithm. Their algorithm may become available in **gRain** in the future.

Acknowledgments

This work was supported by the Danish National Advanced Technology Foundation through the ILSORM project. Peter Green and Steffen Lauritzen are greatly acknowledged for inspiration to the **gRain** package and Vanessa Didelez for valuable comments and for testing **gRain**.

References

AT&T Research (2009). ***Graphviz*** *Version 2.20.3*. URL http://www.graphviz.org/.

Cowell RG, Dawid AP, Lauritzen SL, Spiegelhalter DJ (1999). *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal, Complex Systems*, **1695**. URL http://igraph.sf.net/.

Dawid AP (1992). "Applications of a General Propagation Algorithm for Probabilistic Expert Systems." *Statistics and Computing*, **2**, 25–36.

Gentry J, Long L, Gentleman R, Seth, Hahne F, Sarkar D, Hansen K (2012). ***Rgraphviz:*** *Provides Plotting Capabilities for R Graph Objects*. R package version 1.32.0, URL http://www.bioconductor.org/packages/release/bioc/html/Rgraphviz.html.

Green PJ (2009). ***GRAPPA*** *– A Suite of Functions in R for Probability Propagation in Discrete Graphical Models*. Version 0.3, URL http://www.stats.bris.ac.uk/~peter/Grappa/.

Højsgaard S (2012a). ***gRain:*** *Graphical Independence Networks*. R package version 0.8.12, URL http://CRAN.R-project.org/package=gRain.

Højsgaard S (2012b). ***gRim:*** *Graphical Interaction Models*. R package version 0.1.6, URL http://CRAN.R-project.org/package=gRim.

Højsgaard S, Edwards D, Lauritzen S (2012). *Graphical Models with R*. Springer-Verlag, New York.

Højsgaard S, Thiesson B (1995). "BIFROST – Block Recursive Models Induced from Relevant Knowledge, Observations, and Statistical Techniques." *Journal of Computational Statistics and Data Analysis*, **19**, 155–175.

HUGIN Expert A/S (2011). ***HUGIN** Version 7.5*. Aalborg, Denmark. URL http://www.hugin.com/.

Jensen FV, Olesen KG, Andersen SK (1990). "An Algebra for Bayesian Belief Universes for Knowledge-Based Systems." *Networks*, **20**, 637–659.

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.

Konis K (2011). ***RHugin***. R package version 7.5-6, URL http://RHugin.R-Forge.R-project.org.

Lauritzen SL (1996). *Graphical Models*. Oxford University Press, Oxford.

Lauritzen SL, Spiegelhalter D (1988). "Local Computations with Probabilities on Graphical Structures and their Application to Expert Systems." *Journal of the Royal Statistical Society B*, **50**(2), 157–224.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Tamayo P, Cho YJ, Tsherniak A, Greulich H, Ambrogio L, van Meeteren NS, Zhou T, Buxton A, Kool M, Meyerson M, Pomeroy SL, Mesirov JP (2011). "Predicting Relapse in Medulloblastoma Patients by Integrating Evidence from Clinical and Genomic Features." *Journal of Clinical Oncology*, **29**(11).

Affiliation:

Søren Højsgaard
Department of Mathematical Sciences
Aalborg University
Fredrik Bajers Vej 7G
9220 Aalborg Ø, Denmark
E-mail: sorenh@math.aau.dk
URL: http://people.math.aau.dk/~sorenh/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 46, Issue 10 *Submitted:* 2009-11-20
March 2012 *Accepted:* 2012-02-06