deal: A Package for Learning Bayesian Networks

Susanne G. Bøttcher
Dept. of Mathematical Sciences
Aalborg University
Fr. Bajers Vej 7G
9220 Aalborg, Denmark
alma@math.auc.dk

Claus Dethlefsen
Dept. of Mathematical Sciences
Aalborg University
Fr. Bajers Vej 7G
9220 Aalborg, Denmark
dethlef@math.auc.dk

Abstract

deal is a software package for use with R. It includes several methods for analysing data using Bayesian networks with variables of discrete and/or continuous types but restricted to conditionally Gaussian networks. Construction of priors for network parameters is supported and their parameters can be learned from data using conjugate updating. The network score is used as a metric to learn the structure of the network and forms the basis of a heuristic search strategy. deal has an interface to Hugin.

1 Introduction

A Bayesian network is a graphical model that encodes the joint probability distribution for a set of random variables. Bayesian networks are treated in *e.g.* Cowell et al. (1999) and have found application within many fields, see Lauritzen (2003) for a recent overview.

Here we consider Bayesian networks with mixed variables, *i.e.* the random variables in a network can be of both discrete and continuous types. A method for learning the parameters and structure of such Bayesian networks has recently been described by Bøttcher (2001). We have developed a package called deal, written in R (Ihaka and Gentleman, 1996), which provides these methods for learning Bayesian networks. In particular, the package includes procedures for defining priors, estimating parameters, calculating network scores, performing heuristic search as well as simulating data sets with a given dependency structure. Figure 1 gives an overview of the functionality in deal. The package can be downloaded from the Comprehensive R Archive Network (CRAN) http://cran.R-project.org/ and may be used freely for non-commercial purposes.

In Section 2 we define Bayesian networks for mixed variables. To learn a Bayesian network, the user needs to supply a training data set and represent any prior knowledge available as a Bayesian network. Section 3 shows how to specify the training data set in deal and Section 4 discusses how to specify a Bayesian network in terms of a Directed Acyclic Graph (DAG) and the local probability distributions.

deal uses the prior Bayesian network to deduce prior distributions for all parameters in the model. Then, this is combined with the training data to yield posterior distributions of the parameters. The parameter learning procedure is treated in Section 5.

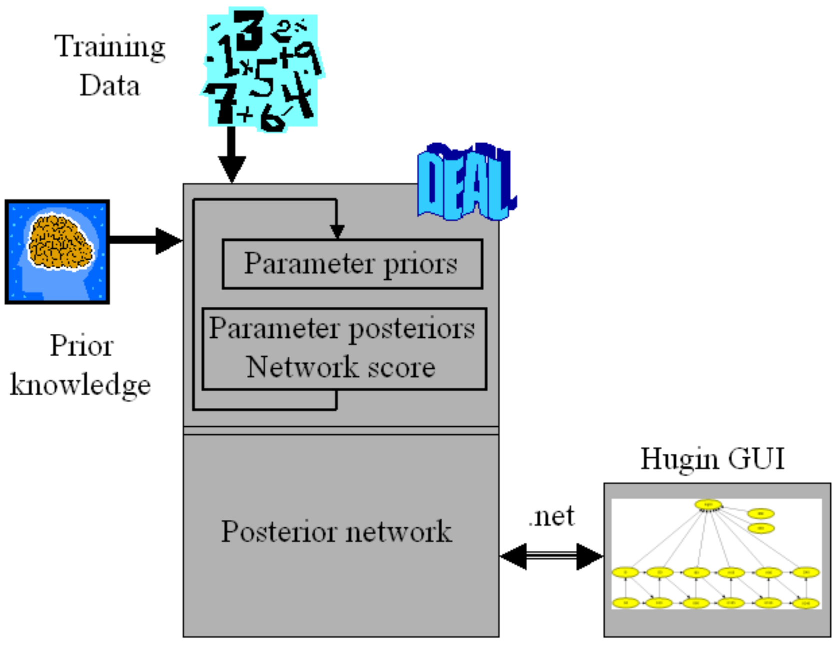

Figure 1: From prior knowledge and training data, a posterior network is produced by `deal`. The network may be transferred to Hugin for further inference.

Section 6 describes how to learn the structure of the network. A network score is calculated and a search strategy is employed to find the network with the highest score. This network gives the best representation of data and we call it the *posterior network*.

Section 7 describes how to transfer the posterior network to Hugin (`http://www.hugin.com`). The Hugin graphical user interface (GUI) can then be used for further inference in the posterior network.

In the appendix we provide manual pages for the main functions in `deal`.

2 Bayesian Networks

Let $D = (V, E)$ be a Directed Acyclic Graph (DAG), where V is a finite set of nodes and E is a finite set of directed edges (arrows) between the nodes. The DAG defines the structure of the Bayesian network.

To each node $v \in V$ in the graph corresponds a random variable X_v. The set of variables associated with the graph D is then $X = (X_v)_{v \in V}$. Often, we do not distinguish between a variable X_v and the corresponding node v. To each node v with parents $\mathrm{pa}(v)$ a local probability distribution, $p(x_v|x_{\mathrm{pa}(v)})$, is attached. The set of local probability distributions for all variables in the network is $\mathcal{P}$.

A Bayesian network for a set of random variables X is the pair $(D, \mathcal{P})$.

The possible lack of directed edges in D encodes conditional independencies between the random variables X through the factorization of the joint probability distribution,

$$p(x) = \prod_{v \in V} p\big(x_v | x_{\mathrm{pa}(v)}\big).$$

Here, we allow Bayesian networks with both discrete and continuous variables, as treated in Lauritzen (1992), so the set of nodes V is given by $V = \Delta \cup \Gamma$, where Δ and Γ are the sets of discrete and continuous nodes, respectively. The set of variables X can then be denoted $X = (X_v)_{v \in V} = (I, Y) = ((I_\delta)_{\delta \in \Delta}, (Y_\gamma)_{\gamma \in \Gamma})$, where I and Y are the sets of discrete and continuous variables, respectively. For a discrete variable, δ, we let $\mathcal{I}_\delta$ denote the set of levels.

To ensure availability of exact local computation methods, we do not allow discrete variables to have continuous parents. The joint probability distribution then factorizes into a discrete part and a mixed part, so

$$p(x) = p(i, y) = \prod_{\delta \in \Delta} p\big(i_\delta | i_{\mathrm{pa}(\delta)}\big) \prod_{\gamma \in \Gamma} p\big(y_\gamma | i_{\mathrm{pa}(\gamma)}, y_{\mathrm{pa}(\gamma)}\big).$$

3 Data Structure

`deal` expects data as specified in a data frame which is a standard data structure in R. For example, standard ASCII data files with one column per variable and one line per observation can be read using `read.table()` which returns a data frame.

The rats example in Table 1 was analysed in Morrison (1976) and Edwards (1995). The data set is from a hypothetical drug trial, where the weight losses of male and female rats under three different drug treatments have been measured after one and two weeks.

Sex	Drug	W1	W2
M	D1	5	6
M	D1	7	6
M	D1	9	9
M	D1	5	4
M	D2	9	12
M	D2	7	7
M	D2	7	6
M	D2	6	8
M	D3	14	11
M	D3	21	15
M	D3	12	10
M	D3	17	12
F	D1	7	10
F	D1	8	10
F	D1	6	6
F	D1	9	7
F	D2	7	6
F	D2	10	13
F	D2	6	9
F	D2	8	7
F	D3	14	9
F	D3	14	8
F	D3	16	12
F	D3	10	5

Table 1: An example data file, *rats.dat.*

The data are loaded into a data frame `rats.df` by the following command

```
rats.df <- read.table("rats.dat",header=TRUE)
```

Before continuing, it is essential that the column variables have the correct types. Discrete variables should be specified as *factors* and continuous variables as *numeric*. To alter the type of a variable so that it is regarded as a discrete variable, use the `factor()` function (standard in R). In the rats example, `Sex` and `Drug` are interpreted to be factors by `read.table`, and thus no changes are necessary.

We assume that we have observed complete data which means that no `NA`'s are present in the data frame.

4 Specification of a Bayesian Network

As described in Section 2, a Bayesian network is specified by a DAG and a set of local probability distributions. In this section we will show how to specify these terms in `deal`.

4.1 The Network Class and Associated Methods

In `deal`, a Bayesian network is represented as an object of class `network`. The network object is a list of properties that are added or changed by the methods described in following sections. A network is generated by the following command

```
rats <- network(rats.df)
```

and by default it is set to the empty network (the network without any arrows).

If the option `specifygraph` is set, a point and click graphical interface allows the user to insert and delete arrows until the requested DAG is obtained.

```
rats  <- network(rats.df,specifygraph=TRUE)
```

A plot of the network (see Figure 2) is generated by

```
plot(rats)
```

Note that discrete nodes are grey and continuous nodes are white.

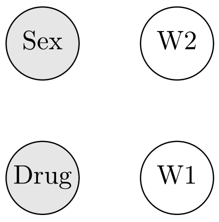

Figure 2: Graphical representation of the rats network.

The primary property of a network is the list of nodes, in the example: `nodes(rats)`. Each entry in the list is an object of class `node` representing a node in the graph, which includes information associated with the node. Several methods for the network class operate by applying an appropriate method for one or more nodes in the list of nodes. The nodes appear in the node list in the same order as in the data frame used to create the network object.

It is possible to access the individual nodes in a network by referring either to their index (the column number in the data frame) or to their name:

```
rats.nd <- nodes(rats)# the list of nodes
rats.nd[[1]]          # the first node
rats.nd$Drug          # the node ‘‘Drug’’
```

A collection of networks can be represented in an object of class `networkfamily`, which has associated `print()` and `plot()` functions.

4.2 Specification of the Probability Distributions

The joint distribution of the random variables in a network in `deal` is a Conditional Gaussian (CG) distribution.

For discrete nodes, this means that the local probability distributions are unrestricted discrete distributions. We parameterize this as

$$\theta_{i_\delta|i_{\mathrm{pa}(\delta)}} = p\big(i_\delta|i_{\mathrm{pa}(\delta)}, \theta_{\delta|i_{\mathrm{pa}(\delta)}}\big),$$

where $\theta_{\delta|i_{\mathrm{pa}(\delta)}} = (\theta_{i_\delta|i_{\mathrm{pa}(\delta)}})_{i_\delta \in \mathcal{I}_\delta}$. The parameters fulfil $\sum_{i_\delta \in \mathcal{I}_\delta} \theta_{i_\delta|i_{\mathrm{pa}(\delta)}} = 1$ and $0 \leq \theta_{i_\delta|i_{\mathrm{pa}(\delta)}} \leq 1$.

For continuous nodes, the local probability distributions are Gaussian linear regressions on the continuous parents with parameters depending on the configuration of the discrete parents. We parameterize this as

$$\theta_{\gamma|i_{\mathrm{pa}(\gamma)}} = \big(m_{\gamma|i_{\mathrm{pa}(\gamma)}}, \beta_{\gamma|i_{\mathrm{pa}(\gamma)}}, \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}}\big),$$

so that

$$\big(Y_\gamma|i_{\mathrm{pa}(\gamma)}, y_{\mathrm{pa}(\gamma)}, \theta_{\gamma|i_{\mathrm{pa}(\gamma)}}\big) \sim \mathcal{N}\big(m_{\gamma|i_{\mathrm{pa}(\gamma)}} + y_{\mathrm{pa}(\gamma)}\beta_{\gamma|i_{\mathrm{pa}(\gamma)}}, \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}}\big).$$

A suggestion for the local probability distributions is generated and attached to each node as the property `prob`. The suggestion can then be edited afterwards.

For a discrete variable δ, the suggested local probability distribution $p(i_\delta|i_{\mathrm{pa}(\delta)})$ is taken to be uniform over the levels for each parent configuration, *i.e.*

$$p(i_\delta|i_{\mathrm{pa}(\delta)}) = 1/\mathcal{I}_\delta.$$

Define $z_{\mathrm{pa}(\gamma)} = (1, y_{\mathrm{pa}(\gamma)})$ and let $\eta_{\gamma|i_{\mathrm{pa}(\gamma)}} = (m_{\gamma|i_{\mathrm{pa}(\gamma)}}, \beta_{\gamma|i_{\mathrm{pa}(\gamma)}})$, where $m_{\gamma|i_{\mathrm{pa}(\gamma)}}$ is the intercept and $\beta_{\gamma|i_{\mathrm{pa}(\gamma)}}$ is the vector of coefficients. For a continuous variable γ, the suggested local probability distribution $\mathcal{N}(z_{\mathrm{pa}(\gamma)}\eta_{\gamma|i_{\mathrm{pa}(\gamma)}}, \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}})$ is determined as a regression on the continuous parents for each configuration of the discrete parents.

The `prob` property for discrete nodes is a multiway array with the node itself occupying the first dimension and the parents each occupying one dimension. For continuous nodes, $\sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}}$ and $\eta_{\gamma|i_{\mathrm{pa}(\gamma)}}$ are stored in a matrix with one row for each configuration of the discrete variables. The first column contains $\sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}}$ and the remaining columns $\eta_{\gamma|i_{\mathrm{pa}(\gamma)}}$.

It is possible to inspect the suggested local probability distributions by setting the option `inspectprob` as

```
rats  <- network(rats.df,inspectprob=TRUE)
```

This gives a graphical way of inspecting the local probability distribution by clicking on the nodes. Then, it is possible to adjust the local distributions, *e.g.*

```
nd <- nodes(rats)
nd$Sex$prob <- c(0.6, 0.4)
nd$W1$prob  <- c(10, 0)
nd$W2$prob  <- c(10, 0, 1) # if eg. W2|W1
nodes(rats) <- nd
```

4.3 The Joint Distribution

We now show how the joint probability distribution of a network can be calculated from the local probability distributions.

For the discrete part of the network, the joint probability distribution is found as

$$p(i) = \prod_{\delta \in \Delta} p\big(i_\delta | i_{\mathrm{pa}(\delta)}\big).$$

For continuous variables, the joint distribution $\mathcal{N}(M_i, \Sigma_i)$ is determined for each configuration of the discrete variables by applying the following sequential algorithm (see Shachter and Kenley (1989)).

The order is determined so that the joint distribution of the parents have already been determined for the current node. For notational convenience, we skip the index i and determine the joint distribution of node γ and all previously processed nodes, p. From the prior network, we have given $\eta_{\gamma|\,\mathrm{pa}(\gamma)} = (m_{\gamma|\,\mathrm{pa}(\gamma)}, \beta_{\gamma|\,\mathrm{pa}(\gamma)})$ and $\sigma^2_{\gamma|\,\mathrm{pa}(\gamma)}$. Previously evaluated are M_p and Σ_p. Now, the covariance is given by

$$\Sigma_{\gamma,p} = \Sigma_p \; \beta_{\gamma|p},$$

where $\beta_{\gamma|p}$ is a column vector of the regression coefficients given all previously evaluated nodes. All coefficients are zero, except the coefficients, $\beta_{\gamma|\,\mathrm{pa}(\gamma)}$, corresponding to the parents of the node. The variance and mean are then given by

$$\begin{aligned} \Sigma_\gamma &= \sigma^2_{\gamma|\,\mathrm{pa}(\gamma)} + \Sigma_{\gamma,p} \; \beta_{\gamma|p} \\ M_\gamma &= m_{\gamma|\,\mathrm{pa}(\gamma)} + \beta^\top_{\gamma|p} \; M_p. \end{aligned}$$

In `deal`, we can assess these quantities by

```
rats.j <- jointprior(rats)
```

and inspect the properties `jointmu`, containing M_i, `jointsigma`, containing Σ_i, and `jointalpha`. The discrete part, $p(i)$, is not returned directly but may be deduced from `rats.j$jointalpha` by division by `sum(rats.j$jointalpha)`.

5 Parameter Learning

In the previous section we showed how to specify a Bayesian network, *i.e.* a DAG and the local probability distributions. In this section we will show how to estimate the parameters in the local probability distributions from data. The first sections present the theory behind the learning procedure and the last section shows how it is done in `deal`.

5.1 The Bayesian Approach

To estimate the parameters in the network, we use the Bayesian approach. We encode our uncertainty about parameters θ in a prior distribution $p(\theta)$, use data d to update this distribution, and hereby obtain the posterior distribution $p(\theta|d)$ by using Bayes' theorem,

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}, \qquad \theta \in \Theta. \tag{1}$$

Here Θ is the parameter space, d is a random sample from the probability distribution $p(x|\theta)$ and $p(d|\theta)$ is the joint probability distribution of d, also called the likelihood of θ. We refer to this as *parameter learning* or just *learning.*

In `deal`, we assume that the parameters associated with one variable are independent of the parameters associated with the other variables and, in addition, that the parameters are independent for each configuration of the discrete parents, *i.e.*

$$p(\theta) = \prod_{\delta\in\Delta} \prod_{i_{\text{pa}(\delta)}\in\mathcal{I}_{\text{pa}(\delta)}} p(\theta_{\delta|i_{\text{pa}(\delta)}}) \prod_{\gamma\in\Gamma} \prod_{i_{\text{pa}(\gamma)}\in\mathcal{I}_{\text{pa}(\gamma)}} p(\theta_{\gamma|i_{\text{pa}(\gamma)}}), \tag{2}$$

We refer to (2) as *parameter independence.* Further, as we have assumed complete data, the parameters stay independent given data, see Bøttcher (2001). This means that we can learn the parameters of a node independently of the parameters of the other nodes, *i.e.* we update the *local parameter prior* $p(\theta_{v|i_{\text{pa}(v)}})$ for each node v and each configuration of the discrete parents.

As local prior parameter distributions we use the Dirichlet distribution for the discrete variables and the Gaussian inverse-Gamma distribution for the continuous variables. These distributions are conjugate to observations from the respective distributions and this ensures simple calculations of the posterior distributions.

In the next section we present an automated procedure for specifying the local parameter priors associated with any possible DAG. The procedure is called the *master prior procedure.* For the mixed case it is treated in Bøttcher (2001), for the purely discrete and the purely continuous cases it is treated in Heckerman et al. (1995) and Geiger and Heckerman (1994), respectively.

5.2 The Master Prior Procedure

In the following sections we will show how to deduce and update the local prior parameter distributions for discrete and continuous nodes, respectively. Here, we will summarize the steps in the master prior procedure.

The idea is that from a given Bayesian network we can deduce parameter priors for any possible DAG. The user just has to specify the Bayesian network as he believes it to be. We call this network a *prior Bayesian network.*

1. Specify a prior Bayesian network, *i.e.* a prior DAG (Section 4.1) and prior local probability distributions (Section 4.2). Calculate the joint prior distribution (Section 4.3).
2. From this joint prior distribution, the marginal distribution of all parameters in the family consisting of the node and its parents can be determined. We call this the *master prior.*
3. The local parameter priors are now determined by conditioning in the master prior distribution.

This procedure ensures parameter independence. Further, it has the property that if a node has the same set of parents in two different networks, then the local parameter prior for this node will be the same in the two networks. Therefore, we only have to deduce the local parameter prior for a node given the same set of parents once. This property is called *parameter modularity.*

5.3 Discrete Nodes

We will now show how to find the local parameter priors for the discrete nodes. Recall that the local probability distributions are unrestricted discrete distributions defined as in Section 4.2.

5.3.1 Master Prior

Let $\Psi = (\Psi_i)_{i \in \mathcal{I}}$ be the parameters for the joint distribution of the discrete variables. The joint prior parameter distribution is assumed to be a Dirichlet distribution

$$p(\Psi) \sim \mathcal{D}(\alpha),$$

with hyperparameters $\alpha = (\alpha_i)_{i \in \mathcal{I}}$. To specify this Dirichlet distribution, we need to specify these hyperparameters.

Consider the following relation for the Dirichlet distribution,

$$p(i) = \mathbb{E}(\Psi_i) = \frac{\alpha_i}{N},$$

with $N = \sum_{i \in \mathcal{I}} \alpha_i$. Now we use the probabilities in the prior network as an estimate of $\mathbb{E}(\Psi_i)$, so we only need to determine N in order to calculate the parameters α_i.

We determine N by using the notion of an imaginary data base. We imagine that we have a data base of cases, from which we have updated the distribution of Ψ out of total ignorance. The *imaginary sample size* of this imaginary data base is thus N. It expresses how much confidence we have in the (in)dependencies expressed in the prior network, see Heckerman et al. (1995).

We use this joint distribution to deduce the master prior distribution of the family $A = \delta \cup \text{pa}(\delta)$. Let

$$\alpha_{i_A} = \sum_{j : j_A = i_A} \alpha_j,$$

and let $\alpha_A = (\alpha_{i_A})_{i_A \in \mathcal{I}_A}$. Then the marginal distribution of Ψ_A is Dirichlet, $p(\Psi_A) \sim \mathcal{D}(\alpha_A)$. This is the master prior in the discrete case.

5.3.2 Local Parameter Prior

From the master prior, we calculate the conditional distribution $\Psi_{\delta | i_{\text{pa}(\delta)}} = \theta_{\delta | i_{\text{pa}(\delta)}}$ which is the local parameter prior in the discrete case. Then,

$$\alpha_{i_\delta | i_{\text{pa}(\delta)}} = \alpha_{i_A},$$

$$\alpha_{\delta | i_{\text{pa}(\delta)}} = \big(\alpha_{i_\delta | i_{\text{pa}(\delta)}}\big)_{i_\delta \in \mathcal{I}_\delta},$$

$$\theta_{\delta | i_{\text{pa}(\delta)}} | \alpha_{i_\delta | i_{\text{pa}(\delta)}} \sim \mathcal{D}\big(\alpha_{\delta | i_{\text{pa}(\delta)}}\big).$$

5.3.3 Local Parameter Posterior

Let $n_{\delta|i_{\mathrm{pa}(\delta)}}$ be the number of cases observed with the particular parent configuration in the data base and let n be the total number of observations.

Then, the posterior parameters $\alpha'_{\delta|i_{\mathrm{pa}(\delta)}}$ are given by

$$\alpha'_{\delta|i_{\mathrm{pa}(\delta)}} = \alpha_{\delta|i_{\mathrm{pa}(\delta)}} + n_{\delta|i_{\mathrm{pa}(\delta)}}.$$

5.4 Continuous Nodes

We now show how to find the local parameter priors for the continuous nodes. Recall that the local probability distributions are normal distributions defined as in Section 4.2.

5.4.1 Master Prior

Bøttcher (2001) derived this procedure in the mixed case. For a configuration i of the discrete variables we let $\nu_i = \rho_i = \alpha_i$, where α_i was determined in Section 5.3.1. Also, $\Phi_i = (\nu_i - 1)\Sigma_i$.

The joint parameter priors are assumed to be distributed as

$$\begin{aligned} p(M_i|\Sigma_i) &= \mathcal{N}\left(\mu_i, \frac{1}{\nu_i}\Sigma_i\right) \\ p(\Sigma_i) &= \mathcal{IW}(\rho_i, \Phi_i), \end{aligned}$$

where $\mathcal{IW}$ is the inverse Wishart distribution.

However, since the marginal distribution of a CG distribution is not necessarily a CG distribution, there is no simple way to derive priors for other networks. Instead we use the imaginary data base to derive local master priors.

Define the notation

$$\rho_{i_{A\cap\Delta}} = \sum_{j:j_{A\cap\Delta}=i_{A\cap\Delta}} \rho_j$$

and similarly for $\nu_{i_{A\cap\Delta}}$ and $\Phi_{i_{A\cap\Delta}}$. For the family $A = \gamma \cup \mathrm{pa}(\gamma)$, the local master prior is then found as

$$\begin{aligned} \Sigma_{A\cap\Gamma|i_{A\cap\Delta}} &\sim \mathcal{IW}\big(\rho_{i_{A\cap\Delta}}, \tilde{\Phi}_{A\cap\Gamma|i_{A\cap\Delta}}\big) \\ M_{A\cap\Gamma|i_{A\cap\Delta}}|\Sigma_{A\cap\Gamma|i_{A\cap\Delta}} &\sim \mathcal{N}\left(\bar{\mu}_{A\cap\Gamma|i_{A\cap\Delta}}, \frac{1}{\nu_{i_{A\cap\Delta}}}\Sigma_{A\cap\Gamma|i_{A\cap\Delta}}\right), \end{aligned}$$

where

$$\begin{aligned} \bar{\mu}_{i_{A\cap\Delta}} &= \frac{\sum_{j:j_{A\cap\Delta}=i_{A\cap\Delta}} \mu_j \nu_j}{\nu_{i_{A\cap\Delta}}} \\ \tilde{\Phi}_{A\cap\Gamma|i_{A\cap\Delta}} &= \Phi_{i_{A\cap\Delta}} + \sum_{j:j_{A\cap\Delta}=i_{A\cap\Delta}} \nu_j(\mu_j - \bar{\mu}_{i_{A\cap\Delta}})(\mu_j - \bar{\mu}_{i_{A\cap\Delta}})^\top. \end{aligned}$$

5.4.2 Local Parameter Prior

Using $\mathcal{I}\Gamma$ for the inverse gamma distribution, the local prior parameters, given as

$$\begin{aligned}\big(m_{\gamma|i_{\mathrm{pa}(\gamma)}}, \beta_{\gamma|i_{\mathrm{pa}(\gamma)}} |\, \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}}\big) &\sim \mathcal{N}\big(\mu_{\gamma|i_{\mathrm{pa}(\gamma)}}, \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}} \tau^{-1}_{\gamma|i_{\mathrm{pa}(\gamma)}}\big)\\ \sigma^2_{\gamma|i_{\mathrm{pa}(\gamma)}} &\sim \mathcal{I}\Gamma\bigg(\frac{\rho_{\gamma|i_{\mathrm{pa}(\gamma)}}}{2}, \frac{\phi_{\gamma|i_{\mathrm{pa}(\gamma)}}}{2}\bigg),\end{aligned}$$

are deduced from the local master prior by conditioning as follows. To simplify notation, we ignore all subscripts in the master prior and thus consider the configuration $i_{A\cap\Delta} = i_{\mathrm{pa}(\gamma)}$ and assume that $A \cap \Gamma$ is ordered with γ as the first entry. Write $\mathrm{pa}(\gamma)$ for the continuous parents $\{A \cap \Gamma\} \setminus \{\gamma\}$. Define the partitioning

$$\tilde{\Phi}_{A\cap\Gamma|i_{A\cap\Delta}} = \left[\begin{array}{cc} \tilde{\phi}_\gamma & \tilde{\Phi}_{\gamma,\mathrm{pa}(\gamma)} \\ \tilde{\Phi}_{\mathrm{pa}(\gamma),\gamma} & \tilde{\Phi}_{\mathrm{pa}(\gamma)} \end{array}\right]$$

$$\bar{\mu}_{i_{A\cap\Delta}} = \Big(\bar{\mu}_\gamma \ , \ \bar{\mu}_{\mathrm{pa}(\gamma)}\Big).$$

Then

$$\mu_{\gamma|i_{\mathrm{pa}(\gamma)}} = \Big(\bar{\mu}_\gamma - \tilde{\Phi}_{\gamma,\mathrm{pa}(\gamma)} \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \bar{\mu}_{\mathrm{pa}(\gamma)} \ , \ \tilde{\Phi}_{\gamma,\mathrm{pa}(\gamma)} \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)}\Big)$$

$$\begin{aligned}\phi_{\gamma|i_{\mathrm{pa}(\gamma)}} &= \tilde{\phi}_\gamma - \tilde{\Phi}_{\gamma,\mathrm{pa}(\gamma)} \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \tilde{\Phi}_{\mathrm{pa}(\gamma),\gamma}\\ \rho_{\gamma|i_{\mathrm{pa}(\gamma)}} &= \rho_{i_{A\cap\Delta}} + |\,\mathrm{pa}(\gamma)|\end{aligned}$$

$$\tau_{\gamma|i_{\mathrm{pa}(\gamma)}} = \left(\begin{array}{cc} 1/\nu_{i_{A\cap\Delta}} + \bar{\mu}^\top_{\mathrm{pa}(\gamma)} \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \bar{\mu}_{\mathrm{pa}(\gamma)} & -\bar{\mu}^\top_{\mathrm{pa}(\gamma)} \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \\ -\tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \bar{\mu}_{\mathrm{pa}(\gamma)} & \tilde{\Phi}^{-1}_{\mathrm{pa}(\gamma)} \end{array}\right)^{-1}.$$

5.4.3 Local Parameter Posterior

Define $z^b_{\mathrm{pa}(\gamma)|i_{\mathrm{pa}(\gamma)}}$ as the matrix with n rows and with a column of ones and columns of the observed continuous parents for a given configuration of the discrete parents. Let $y^b_{\gamma|i_{\mathrm{pa}(\gamma)}}$ be the vector of observations of the node γ for a configuration of the discrete parents.

Then, the prior parameters $\tau_{\gamma|i_{\mathrm{pa}(\gamma)}}$, $\mu_{\gamma|i_{\mathrm{pa}(\gamma)}}$, $\rho_{\gamma|i_{\mathrm{pa}(\gamma)}}$, $\phi_{\gamma|i_{\mathrm{pa}(\gamma)}}$ are updated to posterior parameters (denoted with a prime) by the following relations

$$\tau'_{\gamma|i_{\mathrm{pa}(\gamma)}} = \tau_{\gamma|i_{\mathrm{pa}(\gamma)}} + \big(z^b_{\mathrm{pa}(\gamma)|i_{\mathrm{pa}(\gamma)}}\big)^\top z^b_{\mathrm{pa}(\gamma)|i_{\mathrm{pa}(\gamma)}}$$

$$\mu'_{\gamma|i_{\mathrm{pa}(\gamma)}} = \big(\tau'_{\gamma|i_{\mathrm{pa}(\gamma)}}\big)^{-1} \times \Big(\tau_{\gamma|i_{\mathrm{pa}(\gamma)}} \mu_{\gamma|i_{\mathrm{pa}(\gamma)}} + (z^b_{\mathrm{pa}(\gamma)|i_{\mathrm{pa}(\gamma)}})^\top y^b_{\gamma|i_{\mathrm{pa}(\gamma)}}\Big)$$

$$\begin{aligned}\rho'_{\gamma|i_{\mathrm{pa}(\gamma)}} &= \rho_{\gamma|i_{\mathrm{pa}(\gamma)}} + n\\ \phi'_{\gamma|i_{\mathrm{pa}(\gamma)}} &= \phi_{\gamma|i_{\mathrm{pa}(\gamma)}}\\ &\quad + \big(y^b_{\gamma|i_{\mathrm{pa}(\gamma)}} - z^b_{\mathrm{pa}(\gamma)} \mu'_{\gamma|i_{\mathrm{pa}(\gamma)}}\big)^\top y^b_{\gamma|i_{\mathrm{pa}(\gamma)}}\\ &\quad + \big(\mu_{\gamma|i_{\mathrm{pa}(\gamma)}} - \mu'_{\gamma|i_{\mathrm{pa}(\gamma)}}\big)^\top \tau_{\gamma|i_{\mathrm{pa}(\gamma)}} \mu_{\gamma|i_{\mathrm{pa}(\gamma)}}.\end{aligned}$$

5.5 The Learning Procedure in `deal`

Assume that the training data are available in a data frame, `rats.df`, as described in Section 3. Also, assume that the user has specified a Bayesian network to be used as prior network, called `rats`, see Section 4.

The parameters of the joint distribution of the variables in the network are then determined by the function `jointprior()` with the size of the imaginary data base as optional argument. If the size is not specified, `deal` sets the size to a reasonably small value.

```
rats.prior <- jointprior(rats)    ## auto set size of imaginary data base
rats.prior <- jointprior(rats,12) ## set size of imaginary data base to 12
```

The parameters in the object `rats.prior` may be assessed as

```
rats.prior$jointalpha
rats.prior$jointnu
rats.prior$jointrho
rats.prior$jointphi
```

The procedure `learn()` determines the master prior, local parameter priors and local parameter posteriors and may be called on all nodes or just a single node,

```
rats <- learn(rats,rats.df,rats.prior)$nw   ## all nodes
rats <- learn(rats,rats.df,rats.prior,2)$nw ## only node 2
```

The result is attached to each learned node as two properties. These contain the parameters in the local prior distribution and the parameters in the local posterior distribution, respectively. For the node `Sex`, the property are assessed as

```
nodes(rats)$Sex$condprior
nodes(rats)$Sex$condposterior
```

6 Learning the Structure

In this section we will show how to learn the structure of the DAG from data. The section is based on Bøttcher (2001), Heckerman et al. (1995) and Geiger and Heckerman (1994).

6.1 Network Score

As a measure of how well a DAG D represents the conditional independencies between the random variables, we use the relative probability

$$S(D) = p(D, d) = p(d|D)p(D),$$

and refer to it as a *network score*.

The network score factorizes into a discrete part and a mixed part as

$$S(D) = \prod_{\delta\in\Delta} S_\delta(D) \prod_{\gamma\in\Gamma} S_\gamma(D),$$

where $S_\delta(D)$ is the contribution from the discrete node δ and $S_\gamma(D)$ is the contribution from the continuous node γ.

For a discrete node, δ, the score contribution is given by

$$S_\delta(D) = \prod_{i_{\text{pa}(\delta)} \in \mathcal{I}_{\text{pa}(\delta)}} \frac{\Gamma(\alpha_{+\delta|i_{\text{pa}(\delta)}})}{\Gamma(\alpha_{+\delta|i_{\text{pa}(\delta)}} + n_{+\delta|i_{\text{pa}(\delta)}})} \times \prod_{i_\delta \in \mathcal{I}_\delta} \frac{\Gamma(\alpha_{i_\delta|i_{\text{pa}(\delta)}} + n_{i_\delta|i_{\text{pa}(\delta)}})}{\Gamma(\alpha_{i_\delta|i_{\text{pa}(\delta)}})},$$

where $\alpha_{+\delta|i_{\text{pa}(\delta)}} = \sum_{i_\delta \in \mathcal{I}_\delta} \alpha_{i_\delta|i_{\text{pa}(\delta)}}$ and $n_{+\delta|i_{\text{pa}(\delta)}} = \sum_{i_\delta \in \mathcal{I}_\delta} n_{i_\delta|i_{\text{pa}(\delta)}}$.

For a continuous node, γ,

$$\begin{aligned} S_\gamma(D) \quad = \quad & \prod_{i_{\text{pa}(\gamma)} \in \mathcal{I}_{\text{pa}(\gamma)}} \frac{\Gamma\left(\frac{\rho_{\gamma|i_{\text{pa}(\gamma)}}+n}{2}\right)}{\Gamma\left(\frac{\rho_{\gamma|i_{\text{pa}(\gamma)}}}{2}\right)\sqrt{\det(\rho_{\gamma|i_{\text{pa}(\gamma)}} s_{\gamma|i_{\text{pa}(\gamma)}} \pi)}} \times \\ & \left[1 + \frac{1}{\rho_{\gamma|i_{\text{pa}(\gamma)}}} a_{\gamma|i_{\text{pa}(\gamma)}} s^{-1}_{\gamma|i_{\text{pa}(\gamma)}} a^\top_{\gamma|i_{\text{pa}(\gamma)}}\right]^{-\frac{\rho_{\gamma|i_{\text{pa}(\gamma)}}+n}{2}}, \end{aligned}$$

where

$$\begin{aligned} s_{\gamma|i_{\text{pa}(\gamma)}} &= \frac{\phi_{\gamma|i_{\text{pa}(\gamma)}}}{\rho_{\gamma|i_{\text{pa}(\gamma)}}} \left(I + z^b_{\text{pa}(\gamma)} \tau^{-1}_{\gamma|i_{\text{pa}(\gamma)}} \left(z^b_{\text{pa}(\gamma)}\right)^\top\right) \\ a_{\gamma|i_{\text{pa}(\gamma)}} &= y^b_{\gamma|i_{\text{pa}(\gamma)}} - z^b_{\text{pa}(\gamma)|i_{\text{pa}(\gamma)}} \mu_{\gamma|i_{\text{pa}(\gamma)}}. \end{aligned}$$

Note that the network score factorizes into a product over terms involving only one node and its parents. This property is called *decomposability.*

It can be shown that the network scores for two independence equivalent DAGs are equal. This property is called *likelihood equivalence* and it is a property of the master prior procedure.

In `deal` we use, for computational reasons, the logarithm of the network score. The log network score contribution of a node is evaluated whenever the node is learned and the log network score is updated. The results are inspected as

```
rats <- learn(rats,rats.df,rats.prior)$nw
nodes(rats)$Sex$loglik
rats$score                # log network score
```

6.2 Model Search

In principle, we could evaluate the network score for all possible DAGs and indeed this is provided in `deal`

```
allrats <- networkfamily(rats.df,rats,rats.prior)
allrats <- nwfsort(allrats$nw)
```

However, the number of possible DAGs grows more than exponentially with the number of nodes (see Table 2) and, in general, the problem of identifying the network with the highest score is NP-complete (see Chickering (1996)). If the number of random variables in a network is large, it is not computationally possible to calculate the network score for all the possible DAGs. For these situations a strategy for searching for DAGs with high score is needed. In `deal`, the search strategy *greedy search with random restarts*, see Heckerman et al. (1995), is

# nodes	# networks
1	1
2	2–3
3	12–25
4	144–543
5	4800–29281
6	320000–3781503
7	$\approx 56 \cdot 10^6 - 10^9$
8	$\approx 10^{10} - 10^{11}$
9	$\approx 10^{13} - 10^{15}$
10	$\approx 10^{16} - 10^{18}$

Table 2: The (approximate) number of networks for a given number of nodes. Since we do not allow arrows from continuous to discrete nodes, the number of networks for a given number of nodes is given as a lower and upper bound.

implemented. As a way of comparing the network scores for two different DAGs, D and D^*, we use the posterior odds,

$$\frac{p(D|d)}{p(D^*|d)} = \frac{p(D,d)}{p(D^*,d)} = \frac{p(D)}{p(D^*)} \times \frac{p(d|D)}{p(d|D^*)},$$

where $p(D)/p(D^*)$ is the prior odds and $p(d|D)/p(d|D^*)$ is the Bayes factor. At the moment, the only option in `deal` for specifying prior distribution over DAGs is to let all DAGs be equally likely, so the prior odds are always equal to one. Therefore, we use the Bayes factor for comparing two different DAGs.

In greedy search we compare models that differ only by a single arrow, either added, removed or reversed. In these cases, the Bayes factor is especially simple, because of decomposability of the network score.

Greedy search works as follows.

1. Select an initial DAG D_0, from which to start the search.
2. Calculate Bayes factors between D_0 and all possible networks, which differ by only one arrow, that is
   (a) One arrow is added to D_0
   (b) One arrow in D_0 is deleted
   (c) One arrow in D_0 is turned
3. Among all these networks, select the one that increases the Bayes factor the most.
4. If the Bayes factor is not increased, stop the search. Otherwise, let the chosen network be D_0 and repeat from 2.

In `deal`

```
rats.s <- autosearch(rats,rats.df,rats.prior)$nw
```

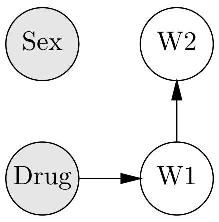

Figure 3: The network with the highest score in the rats example.

returns all tried networks in a greedy search from the initial network `rats`, which may be constructed using `drawnetwork()`.

To manually assess the network score of a network (*e.g.* to use as initial network in a search), use

```
rats <- drawnetwork(rats,rats.df,rats.prior)$nw
```

In the `drawnetwork()` procedure, it is possible to mark (ban) some of the arrows. In the search, `deal` then disregards any DAG which contains any of these arrows, and this reduces the search space.

The search algorithm may also be used with restarts which is implemented in the function `heuristic()`. The initial network is then perturbed according to the parameter `degree` and the search is performed starting with the perturbed network. The process is restarted the number of times specified by the option `restart`. A network family of all visited networks is returned.

```
rats.h <- heuristic(rats,rats.df,rats.prior,restart=10,degree=5)$nw
```

The perturbation of the initial network is done as follows

1. Randomly choose between one of three actions
   (a) Insert an arrow.
   (b) Delete an arrow.
   (c) Turn an arrow.
2. After selection of the action, perform the action according to

   Insert Choose randomly between all possible insertions of one arrow.

   Delete Choose randomly between all possible deletions of one arrow.

   Turn Choose randomly between all possible turns of one arrow.

   If the action is not possible, return the unchanged network.

Perturbation is done automatically in `heuristic()` by calling the function `perturb()`. However, a random graph may also be generated by calling `perturb()` directly

```
rats.rn <- perturb(rats,rats.df,rats.prior,degree=10)$nw
```

6.3 Using Equivalence Relations to Speed up Model Search

In Bøttcher (2003) two types of equivalences are identified and it is shown that no other equivalences exist. Let D_1 and D_1^* be two different networks, that differ by a single arrow between the nodes v and w, with $v \leftarrow w$ in D_1 and $v \not\leftarrow w$ in D_1^*. Further, let D_2 and D_2^* be another two networks that differ by an arrow between v and w.

1. The Bayes factor for testing the arrow from v to w is equivalent to testing this arrow in any other network, where v has the same parents as in D_1.
2. The Bayes factor for testing the arrow from v to w is equivalent to this arrow in the network, where w has the same parents in D_2 as v has in D_1, with the exception that v is also a parent of w in D_2.

We use the first equivalence in all functions that call the learning procedure, including `heuristic()`, `learn()`, `drawnetwork()`, `networkfamily()` and `perturb()`, by maintaining a so-called `trylist`. The `trylist` may be given as input to the functions and is returned in an updated version.

The `trylist` contains a list for each node in the network. The list for a node consists of the result after learning the node for all parent configurations that has previously been tried. When a node is learned after a change in its parent structure, we first look in the `trylist` to see if the node has been learned before with the same parent configuration (Equivalence 1). If the equivalence cannot be used, the node is learned and the result is inserted in the `trylist`. Utilization of Equivalence 2 is not yet implemented in `deal`.

The cost of looking in the `trylist` is smaller than learning a node. Note, however, that the `trylist` must be recalculated if the imaginary data base size is changed or if the data base is changed.

In `deal`, there is support for generating the complete `trylist`, that is, all nodes are learned with all possible parent configurations.

```
rats.tl <- maketrylist(rats,rats.df,rats.prior)
rats.h  <- heuristic(rats,rats.df,rats.prior,trylist=rats.tl)$nw
```

7 Hugin Interface

A network object may be written to a file in the Hugin `.net` language. Hugin (`http://www.hugin.com`) is commercial software for inference in Bayesian networks. Hugin has the ability to learn networks with only discrete networks but cannot learn either purely continuous or mixed networks. `deal` may therefore be used for this purpose and the result can then be transferred to Hugin.

The procedure `savenet()` saves a network to a file. For each node, we use point estimates of the parameters in the local probability distributions.

The `readnet()` procedure reads the network structure from a file but does not, however, read the probability distributions. This is planned to be included in a future version of `deal`.

8 Example

In this section, `deal` is used to analyse a large data set which includes both discrete and continuous variables. The *ksl* data set, included in Badsberg (1995), is from a study measuring health and social characteristics of representative samples of Danish 70-year old people, taken

Node index	Variable	Explanation
1	`Fev`	Forced ejection volume – lung function
2	`Kol`	Cholesterol
3	`Hyp`	Hypertension (no/yes)
4	`BMI`	Body Mass Index
5	`Smok`	Smoking (no/yes)
6	`Alc`	Alcohol consumption (seldom/frequently)
7	`Work`	Working (yes/no)
8	`Sex`	Gender (male/female)
9	`Year`	Survey year (1967/1984)

Table 3: Variables in the *ksl* data set. The variables `Fev`, `Kol`, `BMI` are continuous variables and the rest are discrete variables.

in 1967 and 1984. In total, 1083 cases have been recorded and each case contains observations on nine different variables, see Table 3.

The purpose of our analysis is to find dependency relations between the variables. One interest is to determine which variables influence the presence or absence of hypertension. From a medical viewpoint, it is possible that hypertension is influenced by some of the continuous variables `Fev`, `Kol` and `BMI`. However, in `deal` we do not allow continuous parents of discrete nodes, so we cannot describe such a relation. A way to overcome this problem is to treat `Hyp` as a continuous variable, even though this is obviously not most natural. This is done in the analysis below.

Further, the initial data analysis indicates a transformation of `BMI` into log(`BMI`). With these adjustments, the data set is ready for analysis in `deal`.

First, `deal` is activated and the data are read into a data frame and prepared for analysis.

```
library(deal)  ## invoke DEAL

data(ksl)      ## read data (included in DEAL)
```

The next step in the analysis is to specify a prior Bayesian network. We have no prior knowledge about specific dependency relations, so for simplicity we use the empty DAG as the prior DAG and let the probability distribution of the discrete variables be uniform. The assessment of the probability distribution for the continuous variables is based on data, as described in Section 4.2.

```
## specify prior network
ksl.nw  <- network(ksl)

## make joint prior distribution
ksl.prior <- jointprior(ksl.nw)
```

We do not allow arrows into Sex and Year, as none of the other variables can influence these variables. So we create a ban list which is attached to the network. The ban list is a matrix with two columns. Each row contains the directed edge that is not allowed. The ban list could also have been created interactively using the function `drawnetwork()`.

```
## ban arrows towards Sex and Year
mybanlist <- matrix(c(5,5,6,6,7,7,9,
                      8,9,8,9,8,9,8),ncol=2)
banlist(ksl.nw) <- mybanlist
```

Finally, the parameters in the network are learned and structural learning is initiated using `autosearch()` and `heuristic()`. We use the prior DAG as starting point for the structural search.

```
## learn the initial network
ksl.nw <- learn(ksl.nw,ksl,ksl.prior)$nw

## Do structural search
ksl.search <- autosearch(ksl.nw,ksl,ksl.prior,trace=TRUE)

## perturb 'thebest' and rerun search twice.
ksl.heuristic <- heuristic(ksl.search$nw,ksl,
                           ksl.prior,
                           restart=2,degree=10,
                           trace=TRUE,trylist=ksl.search$trylist)

thebest2 <- ksl.heuristic$nw

savenet(thebest2, "ksl.net")
```

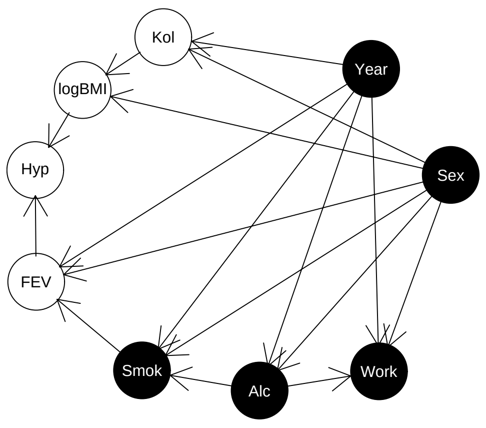

Figure 4: The network with the highest score, log(`score`) $= -15957.91$.

The resulting network `thebest2` is shown in Figure 4 and it is the network with the highest network score among those networks that have been tried through the search.

In the result we see for the discrete variables that `Alc`, `Smok` and `Work` depend directly on `Sex` and `Year`. In addition, `Smok` and `Work` also depend on `Alc`. These two arrows are, however, not causal arrows, as `Smok` $\leftarrow$ `Alc` $\rightarrow$ `Work` in the given DAG represents the same probability distribution as the relations `Smok` $\leftarrow$ `Alc` $\leftarrow$ `Work` and `Smok` $\rightarrow$ `Alc` $\rightarrow$ `Work`, *i.e.* the three DAGs are independence equivalent.

Year and Sex are independent on all variables, as specified in the ban list.

For the continuous variables all the arrows are causal arrows. We see that Fev depends directly on Year, Sex and Smok. So given these variables, Fev is conditional independent on the rest of the variables. Kol depends directly on Year and Sex, and logBMI depends directly on Kol and Sex.

Given logBMI and Fev, the variable Hyp is conditionally independent on the rest of the variables. So according to this study, hypertension can be determined by the body mass index and the lung function forced ejection volume. However, as Hyp is not continuous by nature, other analyses should be performed with Hyp as a discrete variable, *e.g.* a logistic regression with Hyp as a response and the remaning as explanatory variables. Such an analysis indicates that, in addition, Sex and Smok may influence Hyp but otherwise identifies logBMI as the main predictor.

9 Discussion and Future Work

deal is a tool box that adds functionality to R so that Bayesian networks may be used in conjunction with other statistical methods available in R for analysing data. In particular, deal is part of the gR project, which is a newly initiated workgroup with the aim of developing procedures in R for supporting data analysis with graphical models, see http://www.r-project.org/gR.

In addition to methods for analysing networks with either discrete or continuous variables, deal handles networks with mixed variables.

deal has some limitations and we plan to extend the package with the procedures described below. Also, it is the intention that the procedures in deal will eventually be adjusted to the other procedures developed under the gR project.

The methods in deal are only applicable on complete data sets and in the future we would like to incorporate procedures for handling data with missing values and networks with latent variables.

The criteria for comparing the different network structures in deal is the BDe criteria. We intend to also incorporate the Bayesian Information Criteria (BIC) and Akaikes Information Criteria (AIC) and let it be up to the user to decide which criteria to use.

Another possible extension of deal is to incorporate procedures for specifying mixed networks, where the variance in the mixed part of the network does not depend on the discrete parents, but the mean does.

Finally, we are working on an implementation of the greedy equivalence search (GES) algorithm, see Chickering (2002), which is an algorithm for search between equivalence classes. Asymptotically, for the size of the database tending to infinity, this algorithm guarantees that the search terminates with the network with the highest network score.

Acknowledgements

The work has been supported by Novo Nordisk A/S. We thank the reviewer for many helpful comments.

References

J.H. Badsberg. *An Environment for Graphical Models*. PhD thesis, Aalborg University, 1995.

S.G. Bøttcher. Learning Bayesian networks with mixed variables. In *Proceedings of the Eighth International Workshop in Artificial Intelligence and Statistics*, 2001.

S.G. Bøttcher. Learning Bayesian networks with mixed variables. URL `http://www.math.auc.dk/~alma`. Aalborg University, 2003.

David Maxwell Chickering. Learning Bayesian networks is NP-Complete. In D. Fisher and H.J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

D.M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, November 2002.

R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

D. Edwards. *Introduction to Graphical Modelling*. Springer-Verlag, New York, 1995.

D. Geiger and D. Heckerman. Learning Gaussian networks. Technical Report MSR-TR-94-10, Microsoft Research, 1994.

D. Heckerman, D. Geiger, and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 1995.

R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

S.L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 82:1082–1108, 1992.

S.L. Lauritzen. Some modern applications of graphical models. In P.J. Green, N.L. Hjort, and S. Richardson, editors, *Highly Structured Stochastic Systems*. Oxford University Press, 2003.

D.F. Morrison. *Multivariate Statistical Methods*. McGraw-Hill, USA, 1976.

R.D. Shachter and C.R. Kenley. Gaussian influence diagrams. *Management Science*, 35:527–550, 1989.

A Manual Pages for `deal`

`autosearch` *Greedy search*

Description

From initial network, does local perturbations to increase network score.

Usage

```
autosearch(initnw,data,prior=jointprior(network(data)),maxiter=50,
           trylist= vector("list",initnw$n),trace=TRUE,
           timetrace=TRUE,showban=FALSE,saveall=FALSE,removecycles=FALSE)

heuristic(initnw,data,prior=jointprior(network(data)),
          maxiter=100,restart=10,degree=initnw$n,
          trylist= vector("list",initnw$n),trace=TRUE,
          timetrace=TRUE,saveall=FALSE,removecycles=FALSE)

modelstreng(x)
makenw(tb,template)
as.network(x,template)
```

Arguments

`initnw`	an object of class `network`, from which the search is started.
`data`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`maxiter`	an integer, which gives the maximum number of steps in the search algorithm.
`restart`	an integer, which gives the number of times to perturb `initnw` and rerun the search.
`degree`	an integer, which gives the degree of perturbation, see `perturb`.
`trylist`	a list used internally for reusing learning of nodes, see `maketrylist`.
`trace`	a logical. If `TRUE`, plots the accepted networks during search.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.
`showban`	a logical passed to `plot.network`. If `FALSE`, the banned arrows are not shown in the plots (if `trace` is `TRUE`).
`saveall`	obsolete.
`removecycles`	a logical. If `TRUE`, all networks explored in the search is returned, except for networks containing a cycle. If `FALSE`, all networks are returned, including cyclic networks.
`x`	an object of class `network` to be translated into a string.
`tb`	a table output from `autosearch` or `heuristic` in the list property `tabel`. Can be translated into a `networkfamily`.
`template`	an object of class `network` with the same nodes as the networks described in the table `tb`.

Details

In `autosearch`, a list of networks is in each step created with either one arrow added, one arrow deleted or one arrow turned (if a cycle is not generated). The network scores of all the proposal networks are calculated and the network with the highest score is chosen for the next step in the search. If no proposed network has a higher network score than the previous network, the search is terminated. The network with the highest network score is returned, along with a list containing all tried networks (depending on the value of `removecycles`).

`heuristics` restarts by perturbing `initnw` `degree` times and calling `autosearch` again. The number of restarts is given by the option `restart`.

Value

A list with three elements,

`nw`	an object of class `network`, which gives the network with the highest score.
`tabel`	a table with all tried networks. If removecycles is `FALSE`, the networks may contain cycles. The table contains two columns: `model` with a string representation of the model and `score` with the corresponding log network score. The table can be translated to a `networkfamily` using `makenw`.
`trylist`	an updated list used internally for reusing learning of nodes, see `maketrylist`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`perturb`

Examples

```
data(rats)
fit       <- network(rats)
fit.prior <- jointprior(fit,12)
fit       <- learn(fit,rats,fit.prior)$nw
fit       <- insert(fit,2,1,rats,fit.prior)$nw
fit       <- insert(fit,1,3,rats,fit.prior)$nw
hisc      <- autosearch(fit,rats,fit.prior,trace=FALSE)
hisc      <- autosearch(fit,rats,fit.prior,trace=FALSE,removecycles=TRUE) # slower
plot(hisc$nw)

hisc2     <- heuristic(fit,rats,fit.prior,restart=10,trace=FALSE)
plot(hisc2$nw)
print(modelstreng(hisc2$nw))
plot(makenw(hisc2$tabel,fit))
```

`drawnetwork` *Graphical interface for editing networks*

Description

`drawnetwork` allows the user to specify a Bayesian network through a point and click interface.

Usage

```
drawnetwork(nw,df,prior,trylist=vector("list",nw$n),
            unitscale=20,cexscale=8,
            arrowlength=.25,nocalc=FALSE,
            yr=c(0,350),xr=yr,...)

inspectprob(nw,unitscale=20,cexscale=8,
            arrowlength=.25,xr=c(0,350),yr=xr,...)
```

Arguments

`nw`	an object of class `network` to be edited.
`df`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`trylist`	a list used internally for reusing learning of nodes, see `maketrylist`.
`cexscale`	a numeric passed to `plot.network`. Measures the scaled size of text and symbols.
`arrowlength`	a numeric passed to `plot.network`. Measures the length of the edges of the arrowheads.
`nocalc`	a logical. If `TRUE`, no learning procedure is called, see eg. `simulation`.
`unitscale`	a numeric passed to `plot.network`. Scale parameter for chopping off arrow heads.
`xr`	a numeric vector with two components containing the range on x-axis.
`yr`	a numeric vector with two components containing the range on y-axis.
`...`	additional plot arguments, passed to `plot.network`.

Details

To insert an arrow from node 'A' to node 'B', first click node 'A' and then click node 'B'. When the graph is finished, click 'stop'.

To specify that an arrow must not be present, press 'ban' (a toggle) and draw the arrow. This is shown as a red dashed arrow. It is possible to ban both directions between nodes. The ban list is stored with the network in the property `banlist`. It is a matrix with two columns. Each row is the 'from' node index and the 'to' node index, where the indices are the column number in the data frame.

Note that the network score changes as the network is re-learned whenever a change is made (unless `nocalc` is `TRUE`).

`inspectprob` draws the network and makes it possible to inspect the `prob` properties of the nodes by clicking on them. The result is shown in the output window.

Value

A list with two elements,

`nw`	an object of class `network` with the final network.
`trylist`	an updated list used internally for reusing learning of nodes, see `maketrylist`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`network`

Examples

```
data(rats)
rats.nw    <- network(rats)
rats.prior <- jointprior(rats.nw,12)
rats.nw    <- learn(rats.nw,rats,rats.prior)$nw

newrat  <- drawnetwork(rats.nw,rats,rats.prior)$nw
```

`jointprior` *Calculates the joint prior distribution*

Description

Given a network with a `prob` property for each node, derives the joint probability distribution. Then the quantities needed in the local master procedure for finding the local parameter priors are deduced.

Usage

```
jointprior(nw,N=NA,phiprior="bottcher",timetrace=FALSE)
jointdisc(nw,timetrace=FALSE)
jointcont(nw,timetrace=FALSE)
```

Arguments

`nw`	an object of class `network`. Each node must have a `prob` property to describe the local probability distribution. The `prob` property is created using `prob.network`, which is called by the `network` function.
`N`	an integer, which gives the size of the imaginary data base. If this is too small, `NA`'s may be created in the output, resulting in errors in `learn`. If no `N` is given, the procedure tries to set a value as low as possible.
`phiprior`	a string, which specifies how the prior for phi is calculated. Either `phiprior="bottcher"` or `phiprior="heckerman"` can be used.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.

Details

For the discrete part of the network, the joint probability distribution is calculated by multiplying together the local probability distributions. Then, `jointalpha` is determined by multiplying each entry in the joint probability distribution by the size of the imaginary data base `N`.

For the mixed part of the network, for each configuration of the discrete variables, the joint Gaussian distribution of the continuous variables is constructed and represented by `jointmu` (one row for each configuration of the discrete parents) and `jointsigma` (a list of matrices – one for each configuration of the discrete parents). The configurations of the discrete parents are ordered according to `findex`. The algorithm for constructing the joint distribution of the continuous variables is described in Shachter and Kenley (1989).

Then, `jointalpha`, `jointnu`, `jointrho`, `mu` and `jointphi` are deduced. These quantities are later used for deriving local parameter priors.

For each configuration `i` of the discrete variables,

$$\nu_i = \rho_i = \alpha_i$$

and

$$\phi_i = (\nu_i - 1)\Sigma_i$$

if `phiprior="bottcher"`, see Bøttcher(2001) and

$$\phi_i = \nu_i(\rho_i - 2)\Sigma_i/(\nu_i + 1)$$

if `phiprior="heckerman"`, see Heckerman, Geiger and Chickering (1995).

The procedures `jointcont` and `jointdisc` are intended for internal use only.

Value

A list with the following elements,

`jointalpha`	a table used in the local master procedure for discrete variables.
`jointnu`	a table used in the local master procedure for continuous variables.
`jointrho`	a table used in the local master procedure for continuous variables.
`jointmu`	a numeric matrix used in the local master procedure for continuous variables.
`jointsigma`	a list of numeric matrices (not used in further calculations).
`jointphi`	a list of numeric matrices used in the local master procedure for continuous variables.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

References

Bøttcher, S.G. (2001). Learning Bayesian Networks with Mixed Variables, *Artificial Intelligence and Statistics 2001*, Morgan Kaufmann, San Francisco, CA, USA, 149-156.

Heckerman, D., Geiger, D. and Chickering, D. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning **20**: 197-243.

Shachter, R.D. and Kenley, C.R. (1989), Gaussian influence diagrams. Management Science **35**:527-550.

See Also

`network`, `prob.network`

Examples

```
data(rats)
rats.nw    <- network(rats)
rats.prior <- jointprior(rats.nw,12)

savenet(rats.nw,"rats.net")
rats.nw <- readnet("rats.net")
rats.nw <- prob.network(rats.nw,rats)
rats.prior <- jointprior(rats.nw,12)
```

`learn` *Estimation of parameters in the local probability distributions*

Description

Updates the distributions of the parameters in the network, based on a prior network and data. Also, the network score is calculated.

Usage

```
learn (nw, df, prior=jointprior(nw),
               nodelist=1:nw$n,
               trylist=vector("list",nw$n),
               timetrace=FALSE)
learnnode(node,nw,df,prior=jointprior(nw),timetrace=FALSE)

udisclik(node,nw,df)
```

Arguments

`nw`	an object of class `network`.
`df`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`nodelist`	a numeric vector of indices of nodes to be learned.
`trylist`	a list used internally for reusing learning of nodes, see `maketrylist`.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.
`node`	an object of class `node`.

Details

The procedure `learn` determines the master prior, local parameter priors and local parameter posteriors, see Bøttcher (2001). It may be called on all nodes (default) or just a single node.

From the joint prior distribution, the marginal distribution of all parameters in the family consisting of the node and its parents can be determined. This is the master prior, see `localmaster`.

The local parameter priors are now determined by conditioning in the master prior distribution, see `conditional`. The hyperparameters associated with the local parameter prior distribution is attached to each node in the property `condprior`.

Finally, the local parameter posterior distributions are calculated (see `post`) and attached to each node in the property `condposterior`.

A so-called trylist is maintained to speedup the learning process. The trylist consists of a list of matrices for each node. The matrix for a given node holds previously evaluated parent configurations and the corresponding log-likelihood contribution. If a node with a certain parent configuration needs to be learned, it is checked, whether the node has already been learned. The previously learned nodes are given as input in the trylist parameter and is updated in the learning procedure.

When one or more nodes in a network have been learned, the network score is updated and attached to the network in the property `score`.

The learning procedure is called from various functions using the principle, that networks should always be updated with their score. Thus, e.g. `drawnetwork` keeps the network updated when the graph is altered.

The procedures `learnnode` and `udisclik` are intended for internal use.

Value

A list with two elements,

`nw`	an object of class `network`, with the `condposterior` properties updated for the nodes. Also, the property `score` is updated and contains the network score. The contribution to the network score for each node is contained in the property `loglik` for each node.
`trylist`	an updated list used internally for reusing learning of nodes, see `maketrylist`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

References

Bøttcher, S.G. (2001). Learning Bayesian Networks with Mixed Variables, *Artificial Intelligence and Statistics 2001*, Morgan Kaufmann, San Francisco, CA, USA, 149-156.

See Also

`networkfamily`, `jointprior`, `maketrylist`, `network`, `post`

Examples

```
data(rats)
fit       <- network(rats)
fit.prior <- jointprior(fit,12)
fit.learn <- learn(fit,rats,fit.prior,timetrace=TRUE)
fit.nw    <- fit.learn$nw
fit.learn2<- learn(fit,rats,fit.prior,trylist=fit.learn$trylist,timetrace=TRUE)
```

`maketrylist` *Creates the full trylist*

Description

For faster learning, a trylist is maintained as a lookup table for a given parent configuration of a node.

Usage

```
maketrylist(initnw,data,prior=jointprior(network(data)),timetrace=FALSE)
```

Arguments

`initnw`	an object of class `network`, from which the search is started.
`data`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.

Details

This procedure is not intended to be used! For each node in the network, all possible parent configurations are created and learned. The result is called a trylist. To create the full trylist is very time-consuming, and a better choice is to maintain a trylist while searching and indeed this is automatically done. The trylist is given as output to all functions that call the learning procedure and can be given as an argument.

Value

A list with one element per node in the network. In the list, element i is a matrix with two columns: a string with the indices of the parent nodes, separated by ":", and a numeric with the log-likelihood contribution of the node given the parent configuration. Whenever learning is performed of a node given a parent configuration, the trylist is consulted to yield faster learning, especially useful when using `autosearch` or `heuristic`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`networkfamily`, `autosearch` `heuristic`

Examples

```
data(rats)
rats.nw <- network(rats)
rats.pr <- jointprior(rats.nw,12)
rats.nw <- learn(rats.nw,rats,rats.pr)$nw
rats.tr <- maketrylist(rats.nw,rats,rats.pr)

rats.hi <- heuristic(rats.nw,rats,rats.pr,trylist=rats.tr)$nw
```

`network` *Bayesian network data structure*

Description

A Bayesian network is represented as an object of class `network`. Methods for printing and plotting are defined.

Usage

```
network(df,specifygraph=FALSE,inspectprob=FALSE,
        doprob=TRUE,yr=c(0,350),xr=yr)
print(x,filename=NA,condposterior=FALSE,
                         condprior=FALSE,...)
plot (x,arrowlength=.25,
                       notext=FALSE,
                       sscale=7,showban=TRUE,yr=c(0,350),xr=yr,
                       unitscale=20,cexscale=8,...)

prob.network (x,df)

banlist(x)
"banlist<-"(x,value)
```

Arguments

`df`	a data frame, where the columns define the variables. A continuous variable should have type `numeric` and discrete varibles should have type `factor`.
`specifygraph`	a logical. If `TRUE`, provides a call to `drawnetwork` to interactively specify a directed acyclic graph and possibly a ban list (see below).
`inspectprob`	a logical. If `TRUE`, provides a plot of the graph and possibility to inspect the calculated probability distribution by clicking on the nodes.
`doprob`	a logical. If `TRUE`, do not calculate a probability distribution. Used for example in `simulation`.
`x`	an object of class `network`.
`filename`	a string or `NA`. If not `NA`, output is printed to a file.
`condprior`	a logical. If `TRUE`, the conditional prior is printed, see `conditional`.
`condposterior`	a logical. If `TRUE`, the conditional posterior is printed, see `learn`.
`sscale`	a numeric. The nodes are initially placed on a circle with radius `sscale`.
`unitscale`	a numeric. Scale parameter for chopping off arrow heads.
`cexscale`	a numeric. Scale parameter to set the size of the nodes.
`arrowlength`	a numeric containing the length of the arrow heads.
`xr`	a numeric vector with two components containing the range on x-axis.
`yr`	a numeric vector with two components containing the range on y-axis.
`notext`	a logical. If `TRUE`, no text is displayed in the nodes on the plot.
`showban`	a logical. If `TRUE`, banned arrows are shown in red.
`...`	additional plot arguments, passed to `plot.node`.
`value`	a numeric matrix with two columns. Each row contains the indices `i -> j` of arrows that may not be allowed in the directed acyclic graph.

Value

An object of class `network`, which is a list with the following elements (properties),

`nodes`	a list of objects of class `node`. If `doprob` is `TRUE`, the nodes are given the property `prob` which is the initial probability distribution used by `jointprior`.
`n`	an integer containing the number of nodes in the network.
`discrete`	a numeric vector of indices of discrete nodes.
`continuous`	a numeric vector of indices of continuous nodes.
`banlist`	a numeric matrix with two columns. Each row contains the indices `i -> j` of arrows that may not be allowed in the directed acyclic graph.
`score`	a numeric added by `learn` and is the log network score.
`relscore`	a numeric added by `nwfsort` and is the relative network score – compared with the best network in a network family.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`networkfamily`, `node`, `simulation`, `learn`, `drawnetwork`, `jointprior`, `heuristic`, `nwequal`

Examples

```
A <- factor(rep(c("A1","A2"),50))
B <- factor(rep(rep(c("B1","B2"),25),2))
thisnet <- network( data.frame(A,B) )

set.seed(109)
sex     <- gl(2,4,label=c("male","female"))
age     <- gl(2,2,8)
yield   <- rnorm(length(sex))
weight  <- rnorm(length(sex))
mydata  <- data.frame(sex,age,yield,weight)
mynw    <- network(mydata)

# adjust prior probability distribution
mynw$nodes$sex$prob[1:2]    <- c(0.4,0.6)
mynw$nodes$age$prob[1:2]    <- c(0.6,0.4)
mynw$nodes$yield$prob[1:2] <- c(2,0)
mynw$nodes$weight$prob[1:2]<- c(1,0)

print(mynw)
plot(mynw)

prior <- jointprior(mynw)
mynw  <- learn(mynw,mydata,prior)$nw
thebest <- autosearch(mynw,mydata,prior)$nw

print(mynw,condposterior=TRUE)

savenet(mynw,"yield.net")
```

`networkfamily` *Generates and learns all networks for a set of variables.*

Description

Method for generating and learning all networks that are possible for a given set of variables. These may be plotted or printed. Also, functions for sorting according to the network score (see `nwfsort`) and for making a network family unique (see `unique.networkfamily`) are available.

Usage

```
networkfamily(data,nw=network(data), prior=jointprior(nw),
              trylist=vector("list",nw$n), timetrace=TRUE)

print(x,...)
plot(x,layout=<<see below>>,
        cexscale=5,arrowlength=0.1,sscale=7,...)
```

Arguments

`nw`	an object of class `network`. This should be the empty network for the set of variables.
`data`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`trylist`	a list used internally for reusing learning of nodes, see `maketrylist`.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.
`x`	an object of class `networkfamily`.
`layout`	a numeric two dimensional vector with the number of plots in the rows and columns of each plotting page. Default set to `rep(min(1+floor(sqrt(length(x))),5),2)`.
`cexscale`	a numeric. A scaling parameter to set the size of the nodes.
`arrowlength`	a numeric, which gives the length of the arrow heads.
`sscale`	a numeric. The nodes are initially placed on a circle with radius `sscale`.
`...`	additional plot arguments passed to `plot.network`.

Details

`networkfamily` generates and learns all possible networks with the nodes given as in the initial network `nw`. This is done by successively trying to generate the networks with all possible arrows to/from each node (see `addarrows`). If there is a ban list present in `nw` (see `network`), then this is respected, as are the restrictions described in `insert`.

After generation of all possible networks, a test for cycles (see `cycletest`) is performed and only networks with directed acyclic graphs are returned.

Value

The function `networkfamily` returns a list with two components,

`nw`	an object of class `networkfamily`.
`trylist`	an updated list used internally for reusing learning of nodes, see `maketrylist`.

Note

Generating all possible networks can be *very* time consuming!

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`network`, `genlatex`, `heuristic`, `nwfsort`, `unique.networkfamily`, `elementin`, `addarrows`, `cycletest`

Examples

```
data(rats)
allrats <- networkfamily(rats)$nw
plot(allrats)
print(allrats)
```

`node` *Representation of nodes*

Description

An important part of a `network` is the list of nodes. The nodes summarize the local properties of a node, given the parents of the node.

Usage

```
node (idx,parents,type="discrete",name=paste(idx),
                levels=2,levelnames=paste(1:levels),position=c(0,0))
print (x,filename=NA,condposterior=TRUE,condprior=TRUE,...)
plot (x,cexscale=10,notext=FALSE,...)
prob.node (x,nw,df)
nodes(nw)
"nodes<-"(nw,value)
```

Arguments

`x`	an object of class `node`.
`parents`	a numeric vector with indices of the parents of the node.
`idx`	an integer, which gives the index of the node (the column number of the corresponding data frame).
`type`	a string, which gives the type of the node. Either `"discrete"` (for factors) or `"continuous"` (for numeric).
`name`	a string, which gives the name used when plotting and printing. Defaults to the column name in the data frame.
`levels`	an integer. If `type` is `"discrete"`, this is the number of levels for the discrete variable.
`levelnames`	if `type` is `"discrete"`, this is a vector of strings (same length as `levels`) with the names of the levels. If `type` is `"continuous"`, the argument is ignored.
`position`	a numeric vector with coordinates where the node should appear in the plot. Usually set by `network` and `drawnetwork`.
`df`	a data frame, where the columns define the variables. A continuous variable should have type `numeric` and discrete varibles should have type `factor`.
`nw`	an object of class `network`.
`value`	a list of elements of class `node`.
`filename`	a string or `NA`. If not `NA`, output is printed to a file.
`condprior`	a logical. If `TRUE`, the conditional prior is printed, see `conditional`.
`condposterior`	a logical. If `TRUE`, the conditional posterior is printed, see `learn`.
`cexscale`	a numeric. Scale parameter to set the size of the nodes.
`notext`	a logical. If `TRUE`, no text is displayed in the nodes on the plot.
`...`	additional plot arguments.

Details

The operations on a node are typically done when operating on a `network`, so these functions are not to be called directly.

When a network is created with `network`, the nodes in the nodelist are created using the `node` procedure.

Local probability distributions are added as the property `prob` to each node using `prob.node`. If the node is continuous, this is a numeric vector with the conditional variance and the conditional regression coefficients arising from a regression on the continuous parents, using data. If the node has discrete parents, `prob` is a matrix with a row for each configuration of the discrete parents. If the node is discrete, `prob` is a multiway array which gives the conditional probability distribution for each configuration of the discrete parents. The generated `prob` can be replaced to match the prior information available.

`nodes` gives the list of nodes of a network.

Value

An object of class `node`, which is a list with the following elements (properties),

`idx`	an integer. A unique index for this node. It MUST correspond to the column index of the variable in the data frame.
`name`	a string. The printed name of the node.
`type`	a string. Either `"continuous"` or `"discrete"`.
`levels`	an integer. If the node is of type `"discrete"`, this integer is the number of levels of the node.
`levelnames`	if `type` is `"discrete"`, this is a vector of strings (same length as `levels`) with the names of the levels. If `type` is `"continuous"`, the node does not have this property.
`parents`	a vector of indices of the parents to this node. It is best to manage this vector using the `insert` function.
`prob`	a numeric vector, matrix or multiway array, giving the initial probability distribution. If the node is discrete, `prob` is a multiway array. If the node is continuous, `prob` is a matrix with one row for each configuration of the discrete parents, reducing to a vector if the node has no discrete parents.
`condprior`	a list, generated by `conditional` giving the parameter priors deduced from `jointprior` using the master prior procedure (see `localmaster`).
`condposterior`	a list, which gives the parameter posteriors obtained from `learnnode`.
`loglik`	a numeric giving the log likelihood contribution for this node, calculated in `learnnode`.
`simprob`	a numeric vector, matrix or multiway array similar to `prob`, added by `makesimprob` and used by `simulation`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

`numbermixed` *The number of possible networks*

Description

Calculates the number of different directed acyclic graphs for a set of discrete and continuous nodes.

Usage

```
numbermixed(nd,nc)
```

Arguments

`nd`	an integer, which gives the number of discrete nodes.
`nc`	an integer, which gives the number of continuous nodes.

Details

No arrows are allowed from continuous nodes to discrete nodes. Cycles are not allowed. The number of networks is given by Bøttcher (2003), using the result in Robinson (1977).

When nd+nc>15, the procedure is quite slow.

Value

A numeric containing the number of directed acyclic graphs with the given node configuration.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

References

Bøttcher, S.G. (2003). Learning Conditional Gaussian Networks.

Robinson, R.W. (1977). Counting unlabeled acyclic digraphs, *Lecture Notes in Mathematics*, 622: Combinatorial Mathematics.

Examples

```
numbermixed(2,2)
numbermixed(5,10)
```

`nwfsort` *Sorts a list of networks*

Description

According to the `score` property of the networks in a network family, the networks are sorted and the relative score, i.e. the score of a network relative to the highest score, is attached to each network as the `relscore` property.

Usage

```
nwfsort(nwf)
```

Arguments

`nwf`	an object of class `networkfamily`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

`perturb` *Perturbs a network*

Description

Randomly insert/delete/turn arrows to obtain another network.

Usage

```
perturb(nw,data,prior,degree=nw$n,trylist=vector("list",nw$n),
        nocalc=FALSE,timetrace=TRUE)
```

Arguments

`nw`	an object of class `network`, from which arrows are added/removed/turned.
`data`	a data frame used for learning the network, see `network`.
`prior`	a list containing parameter priors, generated by `jointprior`.
`degree`	an integer, which gives the number of attempts to randomly insert/remove/turn an arrow.
`trylist`	a list used internally for reusing learning of nodes, see `maketrylist`.
`nocalc`	a logical. If `TRUE` no learning procedure is called, see eg. `simulation`.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.

Details

Given the initial network, a new network is constructed by randomly choosing an action: remove, turn, add. After the action is chosen, we choose randomly among all possibilities of that action. If there are no possibilites, the unchanged network is returned.

Value

A list with two elements,

`nw`	an object of class `network` with the generated network.
`trylist`	an updated list used internally for reusing learning of nodes, see `maketrylist`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

Examples

```
set.seed(200)
data(rats)
fit       <- network(rats)
fit.prior <- jointprior(fit)
fit       <- learn(fit,rats,fit.prior)$nw
fit.new   <- perturb(fit,rats,fit.prior,degree=10)$nw

data(ksl)
ksl.nw    <- network(ksl)
ksl.rand  <- perturb(ksl.nw,nocalc=TRUE,degree=10)$nw
plot(ksl.rand)
```

`post` *Calculation of parameter posteriors for continuous node*

Description

Learns the parameters and calculates the network score contribution for continuous nodes.

Usage

```
post   (mu,tau,rho,phi,y,z,timetrace=FALSE)
postM  (mu,tau,rho,phi,y,z,timetrace=FALSE)
postc  (mu,tau,rho,phi,y,z,timetrace=FALSE)
postcc (mu,tau,rho,phi,y,z,timetrace=FALSE)
post0  (mu,tau,rho,phi,y,timetrace=FALSE)
postc0c(mu,tau,rho,phi,y,timetrace=FALSE)
```

Arguments

`mu`	a numeric vector of dimension 1 + the number of continuous parents. `mu` is a parameter in the local master, see `conditional`.
`tau`	a numeric matrix, which gives the unscaled precision matrix of regression parameters. Symmetric matrix with number of columns and rows equal to 1 plus number of continous parents.
`rho`	a numeric, which gives a parameter in the distribution of the scale parameter.
`phi`	a numeric, which gives a parameter in the distribution of the scale parameter.
`y`	a numeric vector of observations of the current node.
`z`	a numeric matrix with a column of ones and columns with the observations of the continuous parents.
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.

Details

These functions are called by the learning routines (see `learn`) and is only intended for internal use. In fact, only postc0c and postcc are used for speed reasons. The remaining functions are included for experimental purposes.

post0: posterior for continuous node with 0 parents as batch learning.

postc0c: as post0, but using sequential learning in C.

postc: posterior for continuous node with continuous parents. Sequential learning.

post: as postc, but as batch learning.

postM: as post, but using the Matrix library.

postcc: as postc, but using C.

Value

A list with the following components,

`mu`	numeric vector, giving the posterior mean of the regression parameters.
`tau`	a numeric matrix, which gives the posterior unscaled precision matrix of regression parameters.
`rho`	a numeric, which gives the posterior of a parameter in the distribution of the scale parameter.
`phi`	a numeric, which gives the posterior of a parameter in the distribution of the scale parameter.
`loglik`	a numeric, which gives the log-likelihood contribution to the network score for this node.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`learnnode`, `conditional`

Examples

```
data(rats)
fit        <- network(rats)
fit.prior <- jointprior(fit,12)
W1         <- fit$nodes$W1
W1         <- cond.node(W1,fit,fit.prior)
W1.post    <-  postc0c(W1$condprior[[1]]$mu,
                      W1$condprior[[1]]$tau,
                      W1$condprior[[1]]$rho,
                      W1$condprior[[1]]$phi,
                      rats[,W1$idx])
```

`readnet` *Reads/saves .net file*

Description

Reads/saves a Bayesian network specification in the `.net` language (see `http://developer.hugin.com/documentation/net/`).

Usage

```
readnet(filename)
savenet(nw, filename = "default.net")
```

Arguments

`filename`	a string, which contains the file name to be read.
`nw`	an object of class `network`.

Details

`readnet` reads only the structure of a network, i.e. the directed acyclic graph.

`savenet` exports the `prob` property for each node in the network object along with the network structure defined by the parents of each node.

Value

`readnet` creates an object of class `network` with the nodes specified as in the `.net` file. The network has not been learned and the nodes do not have `prob` properties (see `prob.network`).

`savenet` creates a file.

Note

The call to `readnet(savenet(network))` is *not* the identity function as information is thrown away in both `savenet` and `readnet`.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

See Also

`network`

Examples

```
data(rats)
nw <- network(rats)
savenet(nw,"default.net")
nw2 <- readnet("default.net")
nw2 <- prob.network(nw2,rats)
```

`simulation` *Simulation of data sets with a given dependency structure*

Description

Given a network with nodes having the `simprob` property, `simulation` simulates a data set.

Usage

```
simulation(nw, n=24, file="")
```

Arguments

`nw`	an object of class `network`, where each node has the property `simprob` (see `makesimprob`).
`n`	an integer, which gives the number of cases to simulate.
`file`	a string. If non-empty, the data set is stored there.

Details

The variables are simulated one at a time in an order that ensures that the parents of the node have already been simulated. For discrete variables a multinomial distribution is used and for continuous variables, a Gaussian distribution is used, according to the `simprob` property in each node.

Value

A data frame with one row per case. If a file name is given, a file is created with the data set.

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

Examples

```
A  <- factor(NA,levels=paste("A",1:2,sep=""))
B  <- factor(NA,levels=paste("B",1:3,sep=""))
c1 <- NA
c2 <- NA
df <- data.frame(A,B,c1,c2)

nw <- network(df,doprob=FALSE) # doprob must be FALSE
nw <- makesimprob(nw)          # create simprob properties

set.seed(944)
sim <- simulation(nw,n=100)    # create simulated data frame
```

`unique.networkfamily` *Makes a network family unique.*

Description

Removes networks that are equal or equivalent to networks already in the network family.

Usage

```
unique(x,incomparables=FALSE,equi=FALSE,timetrace=FALSE,epsilon=1e-12,...)
```

Arguments

`x`	an object of class `networkfamily`.
`incomparables`	a logical, but has no effect.
`equi`	a logical. If `TRUE`, also equivalent networks are thrown out (*i.e.* if their score is within `epsilon` from another network).
`timetrace`	a logical. If `TRUE`, prints some timing information on the screen.
`epsilon`	a numeric, which measures how close network scores are allowed to be from each other to be 'equivalent'.
`...`	further arguments (no effect)

Author(s)

Susanne Gammelgaard Bøttcher ⟨alma@math.auc.dk⟩,
Claus Dethlefsen ⟨dethlef@math.auc.dk⟩.

Examples

```
data(rats)
rats.nwf <- networkfamily(rats)
rats.nwf2<- unique(rats.nwf$nw,equi=TRUE)
```