Data Modelling using Additive Bayesian Networks

Fraser Ian Lewis, Marta Pittavino, Reinhard Furrer

Abstract

This vignette describes the **abn** package of R which provides functionality for identifying statistical dependencies in complex data using additive Bayesian network models. This methodology is ideally suited for both univariate - one response variable, and multiple explanatory variables - and multivariate analyses, where in both cases all statistical dependencies between all variables in the data are sought. These models comprise of directed acyclic graphs (DAGs) where each node in the graph comprises a generalized linear model, where model search algorithms are used to identify those DAG structures most supported by the data. Currently implemented are models for data comprising of categorical and/or continuous variables. Further relevant information about **abn** can be found at: www.r-bayesian-networks.org.

Keywords: R, Bayesian Networks, additive models, structure discovery.

1. Introduction

Bayesian network (BN) modeling (Buntine 1991; Heckerman, Geiger, and Chickering 1995; Lauritzen 1996; Jensen 2001) is a form of graphical modeling which attempts to separate out indirect from direct association in complex multivariate data, a process typically referred to as structure discovery (Friedman and Koller 2003). Unlike other widely used multivariate approaches where dimensionality is reduced through exploiting linear combinations of random variables, such as in principal component analysis, graphical modeling does not involve any such dimension reduction. Bayesian networks have been developed for analysing multinomial, multivariate Gaussian or conditionally Gaussian networks (a mix categorical and Gaussian variables). A number of libraries for fitting such BNs are available from CRAN. These types of BN have been constructed to ensure conjugacy, that is, enable posterior distributions for the model parameters and marginal likelihood to be calculated analytically. The purpose of **abn** is to provide a library of functions for more flexible BNs which do not rely on conjugacy, which opens up an extremely rich modeling framework but at some considerable additional computational cost.

Currently **abn** includes functionality for fitting non-conjugate BN models which are multi-dimensional analogues of combinations of Binomial (logistic) and Gaussian regression. It includes also model with Poisson (log) distribution for count data and generalised linear models with random effects (with the previous distributions). It is planned to extend this to include more complex distributions for overdispersed data such a beta-binomial and negative binomial.

The objective in BN modeling structure discovery is to perform a model search on the data to identify an optimal model. Recall that BN models have a vast search space - super-exponential in the number of nodes - and it is generally impossible to determine a globally optimal model. How best to summarize a set of locally optimal networks with different structural features is an open question, and there are a number of widely used and intuitively reasonable possibilities. For example, one option is to conduct a series of heuristic searches and then simply select the best model found (Heckerman *et al.* 1995); alternatively, a single summary network can be constructed using results across many different searches (Hodges, Dai, Xiang, Woolf, Xi, and He 2010; Poon, Lewis, Pond, and Frost 2007). There are obvious pros and cons to either approach and both are common in the literature and provide a good first exploration of the data. For a general non-technical review of BN modeling applied in biology see (Needham, Bradford, Bulpitt, and Westhead 2007). A case study in applying BN models to epidemiological data using the conjugate BN functionality in **abn** can be found in (Lewis, Brulisauer, and Gunn 2011).

In this vignette we consider a series of examples illustrating how to fit different types of models and run different searches and summary analyses to a (synthetic) data set comprising of 250 observations from a joint distribution comprising of 17 categorical and 16 continuous variables which is included as part of the **abn** library. This data set is a single realization from a network of the same structure as that presented in (Lewis *et al.* 2011), which is based on real data and sufficiently complex to provide a realistic example of data mining using Bayesian Network modeling. Another detailed introduction and further relevant case studies about **abn** can be found at: www.r-bayesian-networks.org.

2. Case Study Data

Figure 1 shows the structure of the distribution which generated the data set `var33` included with

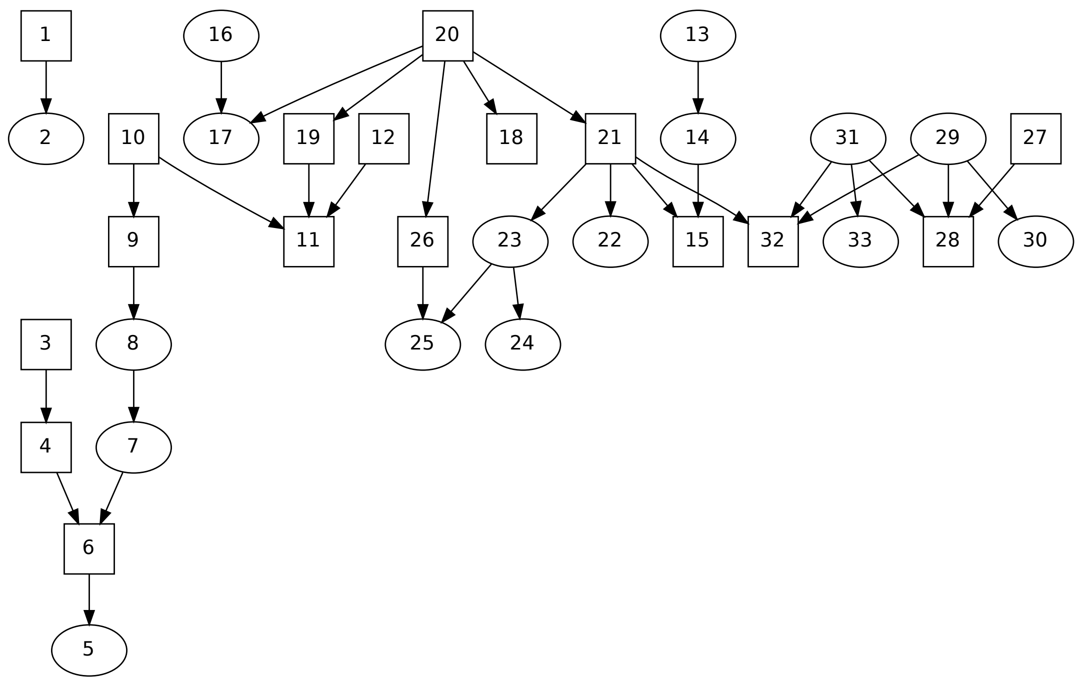

Figure 1: Directed acyclic graph representation of the joint probability distribution which generated data set `var33` which is included with **abn**. The square nodes are categorical (binary) and the oval nodes continuous variables.

abn. This diagram was created using the `tographviz()` function of **abn** (see later examples) which translates the matrix which defines a network - a directed acyclic graph - into a text file of suitable format for processing in Graphviz, where this processing was done outside of R. Graphviz is freely available and operates on most platforms and can be downloaded from www.graphviz.org, there is also an R package which interfaces to Graphviz available from the Bioconductor project (requires an installation of Graphviz).

3. Fitting a single BN model to data

In the next sections we illustrate how to fit a BN model to different kinds of data. The main purpose of BN structure discovery is to estimate the joint dependency structure of the random variables in the available data, and this is achieved by heuristically searching for optimal models and comparing their goodness of fit using Bayes factors. It is assumed that all structures are equally supported in the absence of any data - an uniformative prior on structures - and so comparing Bayes factors collapses to comparing the marginal likelihoods which is done on a log scale. The log marginal likelihood for a BN is typically referred to as the network score.

3.1. Fitting an additive BN model to categorical data

An additive BN model for categorical data can be constructed by considering each individual variable as a logistic regression of the other variables in the data, and hence the network model comprises of many combinations of local logistic regressions (Rijmen 2008). The parameters in this model are the additive terms in a usual logistic regression and independent Gaussian priors are assumed for each covariate. Note that the variables here must all be binary, and so all multinomial variables need to be split into separate binary factors (and added to the original data.frame) in order to form the network model. This is analogous to forming the design matrix in a conventional additive model analysis. Similarly, interaction terms can be added by including appropriate additional columns in the data.frame. In these models the log marginal likelihood (network score) is estimated using Laplace approximations at each node. Hyperparameters for the means and variances in the Gaussian priors are fixed at zero and 1000 respectively, and other values can be given explicitly in the call to `fitabn` but this is not recommended without good reason.

To fit an additive model use `fitabn(data.df,dag.m,data.dists, ...)`. In the following code we fit first the independence model with no arcs and then the same dependence model as above. Turning on `verbose=TRUE` simply gives the individual log marginal likelihoods for each node (n.b. the numbering is that used internally and simply denotes the variables in the data.frame from left to right).

The following code fits a network to the subset of the variables from `var33` which are categorical. In this data these are all binary. Note that all categorical variables should be set as factors.

```
> library( abn, lib="/home/b/martapit/todelete")
> require( abn) # load library
> bin.nodes<-c( 1,3,4,6,9,10,11,12,15,18,19,20,21,26,27,28,32);
> var33.cat<-var33[,bin.nodes]; #categorical nodes only
> mydag<-matrix(c(  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  #v1
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  #v3
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  #v4
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  #v6
+                   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  #v9
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v10
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v11
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v12
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v15
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v18
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v19
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v20
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v21
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v26
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v27
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #v28
+                    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0  #v32
+                 ),byrow=TRUE,ncol=17);
> colnames( mydag)<-rownames( mydag)<-names( var33.cat);#set names
> ## move back to independence model
> mydag["v11","v12"]<-0;mydag["v11","v10"]<-0;mydag["v4","v3"]<-0;
> ## setup distribution list for each categorical node
> mydists.cat<-list( v1 ="binomial", v3 = "binomial",
+        v4 = "binomial",  v6 = "binomial",  v9 = "binomial",
+       v10 = "binomial", v11 = "binomial", v12 = "binomial",
+       v15 = "binomial", v18 = "binomial", v19 = "binomial",
+       v20 = "binomial", v21 = "binomial", v26 = "binomial",
+       v27 = "binomial", v28 = "binomial", v32 = "binomial");
> ind.mod.cat <- fitabn( data.df=var33.cat, dag.m=mydag,
+                       data.dists=mydists.cat, verbose=FALSE);
> ## change to verbose=TRUE if one want to check how change the
> ## score for each individual node
> ind.mod.cat$mlik

[1] -2856.948

> ## network score for a model with conditional independencies
```

The structure of the network definition matrix is where each row is a “child” and each column is its “parents”, where a `1` denotes a parent (or arc) is present. Now lets fit a model with some conditional dependencies, for example where `v11` is conditionally dependent upon `v12` and `v10`, and `v4` is conditionally dependent upon `v3`.

```
> ## now fit the model with some conditional dependencies
> mydag["v11","v12"]<-1;mydag["v11","v10"]<-1;mydag["v4","v3"]<-1;
> dep.mod.cat <- fitabn( data.df=var33.cat, dag.m=mydag,
+                       data.dists=mydists.cat, verbose=FALSE);
> dep.mod.cat$mlik

[1] -2850.081
```

```
> ## network score for a model with conditional dependencies
```

The network score is considerably improved and therefore suggests support for these new structural features. To produce a visual description of the model then we can export to graphviz as follows

```
> tographviz( dag=mydag, data.df=var33.cat, data.dists=mydists.cat,
+             outfile="mydagcat.dot", directed=TRUE);#create file
> # mydagcat.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydagcat.dot -o mydagcat.pdf"
> # or use gedit if on Windows
```

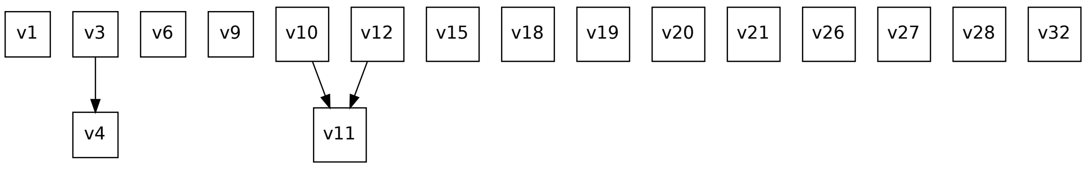

Figure 2: Directed acyclic graph `mydag` created using `tographviz()` and Graphviz

In `tographviz()` the `data.df` argument is used to determine whether the variable is a factor or not, where factors are displayed as squares and non-factors as ovals. To use the full range of visual Graphviz options simply use the file created by `tographviz()` as a starting point and manually edit this in a text editor before running through `dot` or one of the other Graphviz layout processors.

3.2. Fitting an additive BN model to continuous data

We now consider analogous models to those in Section 3.1 but where the network comprises of Gaussian linear regressions rather than logistic regressions. The structure of these models again assumes independent Gaussian priors for each of the coefficients in the additive components for the mean response at each node (with hyper means = 0 and hyper variances = 1000). The Gaussian response distribution is parameterized in terms of precision $(1/\sigma^2)$, and independent Gamma priors are used with shape=0.001 and scale=1/0.001 (where these are as defined in the `rgamma` help page). By default, each variable in the data.frame is standardised to a mean of zero and standard deviation of one, this has no effect on the identification of dependencies between variables.

```
> var33.cts<-var33[,-bin.nodes]; #drop categorical nodes
> mydag<-matrix( 0, 16, 16);
> colnames( mydag)<-rownames( mydag)<-names( var33.cts);#set names
> ## setup distribution list for each continuous node
> mydists.cts<-list( v2 = "gaussian", v5 = "gaussian",
+         v7 = "gaussian", v8 = "gaussian", v13 = "gaussian",
+         v14 = "gaussian", v16 = "gaussian", v17 = "gaussian",
+         v22 = "gaussian", v23 = "gaussian", v24 = "gaussian",
+         v25 = "gaussian", v29 = "gaussian", v30 = "gaussian",
+         v31 = "gaussian", v33 = "gaussian");
> ## now fit the model defined in mydag - full independence
> ind.mod.cts <- fitabn( data.df=var33.cts, dag.m=mydag,
+                 data.dists=mydists.cts, verbose=FALSE);
> ## uses default priors: N(mu=0,var=1000), 1/var=Gamma(0.001,1/0.001)
> ind.mod.cts$mlik
```

```
[1] -5949.52
```

```
> # this is the network score=goodness of fit=log marginal likelihood
```

Now fit a model with conditional independencies, for example

```
> # now fit model with some conditional dependencies let v33
> ## depend on v31, and v24 depend on 23, and v14 depend on v13
> mydag["v33","v31"]<-1;
> mydag["v24","v23"]<-1;
> mydag["v14","v13"]<-1;
> dep.mod.cts <- fitabn( data.df=var33.cts, dag.m=mydag,
+                 data.dists=mydists.cts, verbose=FALSE);
> dep.mod.cts$mlik
```

```
[1] -5704.547
```

```
> # network score for a model with conditional independence
> tographviz( dag=mydag, data.df=var33.cts, data.dists=mydists.cts,
+             outfile="mydagcts.dot", directed=TRUE);#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydagcts.dot -o mydagcts.pdf" or
> # use gedit if on Windows
```

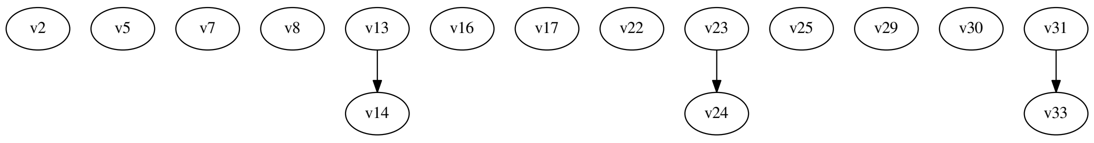

Figure 3: Directed acyclic graph `mydag` for continuous variables only created using `tographviz()` and Graphviz

3.3. Fitting an additive BN model to mixed data

To conclude the fitting of a single pre-specified model to data, e.g. based on expert opinion, we consider an additive BN model which comprises both binary and Gaussian nodes and this comprises of a combination of Binomial (logistic) and Gaussian linear models. Again `fitabn()` is used and the code is almost identical to the previous examples.

```
> mydag<-matrix( 0, 33, 33);
> colnames( mydag)<-rownames( mydag)<-names( var33);#set names
> ## setup distribution list for each mixed node
> mydists.mix<-list( v1 = "binomial", v2 = "gaussian",
+         v3 = "binomial", v4 = "binomial", v5 = "gaussian",
+         v6 = "binomial", v7 = "gaussian", v8 = "gaussian",
+         v9 = "binomial", v10 = "binomial", v11 = "binomial",
+         v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
+         v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
+         v18 = "binomial", v19 = "binomial", v20 = "binomial",
+         v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
+         v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
+         v27 = "binomial", v28 = "binomial", v29 = "gaussian",
+         v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
+         v33 = "gaussian");
> ## now fit the model defined in mydag - full independence
> ind.mod <- fitabn( data.df=var33, dag.m=mydag,
+           data.dists=mydists.mix, verbose=FALSE);
> ind.mod$mlik
```

```
[1] -8806.468
```

```
> # this is the network score with no conditional dependencies
```

We now fit a BN model which has the same structure as the joint distribution used to generate the data and then create a visual graph of this model

```
> # define a model with many independencies
> mydag[2,1]<-1;
> mydag[4,3]<-1;
> mydag[6,4]<-1; mydag[6,7]<-1;
> mydag[5,6]<-1;
> mydag[7,8]<-1;
> mydag[8,9]<-1;
> mydag[9,10]<-1;
> mydag[11,10]<-1; mydag[11,12]<-1; mydag[11,19]<-1;
> mydag[14,13]<-1;
> mydag[17,16]<-1;mydag[17,20]<-1;
> mydag[15,14]<-1; mydag[15,21]<-1;
> mydag[18,20]<-1;
> mydag[19,20]<-1;
> mydag[21,20]<-1;
> mydag[22,21]<-1;
> mydag[23,21]<-1;
> mydag[24,23]<-1;
> mydag[25,23]<-1; mydag[25,26]<-1;
> mydag[26,20]<-1;
> mydag[33,31]<-1;
> mydag[33,31]<-1;
> mydag[32,21]<-1; mydag[32,31]<-1;mydag[32,29]<-1;
> mydag[30,29]<-1;
> mydag[28,27]<-1; mydag[28,29]<-1;mydag[28,31]<-1;
> dep.mod <- fitabn( data.df=var33, dag.m=mydag,
+            data.dists=mydists.mix, verbose=FALSE);
> dep.mod$mlik

[1] -8019.887

> # network score for a model with conditional independence
> tographviz( dag=mydag, data.df=var33, data.dists=mydists.mix,
+             outfile="mydag_all.dot", directed=TRUE);#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf mydag_all.dot -o mydag_all.pdf" or use
> # gedit if on Windows
```

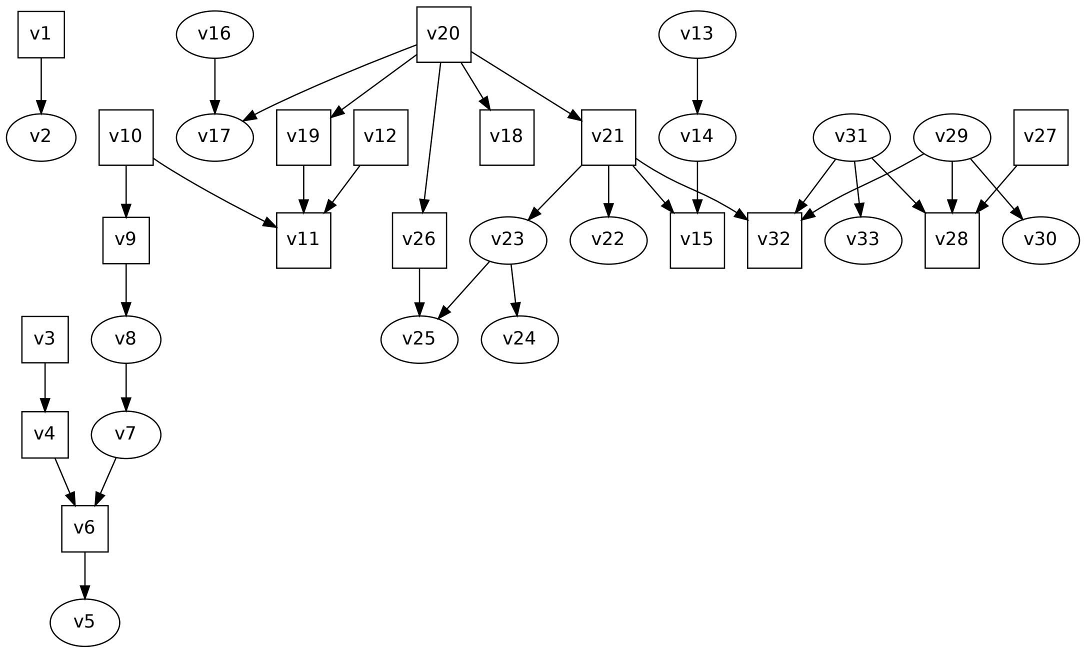

Figure 4: Directed acyclic graph mydag for mixed continuous and discrete variables

3.4. Model fitting validation

In order to validate the additive models for mixed binary and Gaussian models, estimates of the posterior distributions for the model parameters using Laplace approximations were compared with those estimated using Markov chain Monte Carlo. These were always in very close agreement for the range of models and data examined. This is an indirect validation of the Laplace estimate of the network score, e.g. if the posterior densities match closely then this implies that the denominator (the marginal likelihood - network score) must also be accurately estimated, as a "gold standard" estimate of the network score is generally unavailable for such non-conjugate models.

4. Searching for Optimal Models

The key objective of the **abn** library is to enable estimation of statistical dependencies in data comprising of multiple variables - that is, find a DAG which is robust and representative of the dependency structure of the (unknown) stochastic system which generated the observed data. The challenge here is that with such a vast model space it is impossible to enumerate over all possible DAGs, and there may be very many different DAGs with similar goodness of fit. In the next sections we first consider searching for additive (non-conjugate) models.

4.1. Single search for optimal additive BN model from categorical data

To run a single search heuristic use `search.hillclimber()`. This commences from a randomly created DAG which is constructed by randomly adding arcs to an empty network until all possible arcs have been tried. The function `search.hillclimber()` then searches stepwise from the initial random network for an improved structure, where three stepwise operations are possible: i) add an arc; ii) remove and arc; or iii) reverse and arc. The stepwise search is subject to a number of conditions, firstly only moves that do not generate a cycle are permitted, secondly, a parent limit is imposed which fixes the maximum number of parents which each child node can have (arcs go from parent to child), and thirdly it is possible to ban or retain arcs. If provided, `banned.m` is a matrix which defines arcs that are not allowed to be considered in the search process (or in the creation of the initial random network). Similarly, `retain.m` includes arcs which must always be included in any model. It is also possible to specific an explicit starting matrix, `start.m` and if using a retain matrix then `start.m` should contain at least all those arcs present in `retain.m`. Note that only very rudimentary checking is done to make sure that the ban, retain and start networks - if user supplied - are not contradictory.

To improve the computational performance of `search.hillclimber()` a cache of all possible goodness of fit must be built in advance, using the function `buildscorecache()`. Rather than re-calculate the score for each individual node in the network (the overall network score is the product of all the scores for the individual nodes) the score for each unique node found during the search is stored in the cache created by the function `buildscorecache()`.

```
> bin.nodes<-c(1,3,4,6,9,10,11,12,15,18,19,20,21,26,27,28,32);
> var33.cat<-var33[,bin.nodes];#categorical nodes only
> mydag<-matrix( 0, 17, 17);
> colnames(mydag)<-rownames(mydag)<-names(var33.cat);#set names
> ## create banned and retain empty DAGs
> banned.cat<-matrix( 0, 17, 17);
> colnames(banned.cat)<-rownames(banned.cat)<-names(var33.cat);
> retain.cat<-matrix( 0, 17, 17);
> colnames(retain.cat)<-rownames(retain.cat)<-names(var33.cat);
> ## setup distribution list for each categorical node
> mydists.cat<-list( v1 ="binomial", v3 = "binomial",
+        v4 = "binomial",  v6 = "binomial",  v9 = "binomial",
+       v10 = "binomial", v11 = "binomial", v12 = "binomial",
+       v15 = "binomial", v18 = "binomial", v19 = "binomial",
+       v20 = "binomial", v21 = "binomial", v26 = "binomial",
+       v27 = "binomial", v28 = "binomial", v32 = "binomial");
> ## build cache of all the local computations
> ## this information is needed later when running a model search
> mycache.cat<-buildscorecache( data.df=var33.cat,
+              data.dists=mydists.cat, dag.banned=banned.cat,
+              dag.retained=retain.cat, max.parents=1);
```

Running a single search heuristic for an additive BN uses `search.hillclimber()`. It uses a parameter prior specifications (as detailed above). Several additional arguments are available which relate to the numerical routines used in the Laplace approximation to calculate the network score. The defaults appear to work reasonably well in practice and if it is not possible to calculate a robust value for this approximation in any model, for example due to a singular design matrix at one or more nodes, then the model is simply assigned a log network score of $-\infty$ which effectively removes it from the model search.

```
> # Run a single search heuristic for an additive BN
> heur.res.cat<-search.hillclimber( score.cache=mycache.cat,
+               num.searches=1, seed=0, verbose=FALSE,
+               trace=FALSE, timing.on=FALSE);
> # Setting trace=TRUE, the majority consensus network is
> # plotted as the searches progress
```

4.2. Single search for optimal additive BN model for continuous data

As above but for a network of Gaussian nodes.

```
> var33.cts<-var33[,-bin.nodes];#drop categorical nodes
> mydag<-matrix( 0, 16, 16);
> colnames(mydag)<-rownames(mydag)<-names(var33.cts);#set names
> banned.cts<-matrix( 0, 16, 16);
> colnames(banned.cts)<-rownames(banned.cts)<-names(var33.cts);
> retain.cts<-matrix( 0, 16, 16);
> colnames(retain.cts)<-rownames(retain.cts)<-names(var33.cts);
> ## setup distribution list for each continuous node
> mydists.cts<-list( v2 = "gaussian", v5 = "gaussian",
+         v7 = "gaussian", v8 = "gaussian", v13 = "gaussian",
+         v14 = "gaussian", v16 = "gaussian", v17 = "gaussian",
+         v22 = "gaussian", v23 = "gaussian", v24 = "gaussian",
+         v25 = "gaussian", v29 = "gaussian", v30 = "gaussian",
+         v31 = "gaussian", v33 = "gaussian");
> ## build cache of all the local computations
> ## this information is needed later when running a model search
> mycache.cts<-buildscorecache( data.df=var33.cts,
+              data.dists=mydists.cts, dag.banned=banned.cts,
+              dag.retained=retain.cts, max.parents=1);
```

```
> # Run a single search heuristic for an additive BN
> heur.res.cts<-search.hillclimber( score.cache=mycache.cts,
+               num.searches=1, seed=0, verbose=FALSE,
+               trace=FALSE, timing.on=FALSE);
> # Setting trace=TRUE, the majority consensus network is
> # plotted as the searches progress
```

4.3. Single search for optimal additive BN model for mixed data

Model searching for mixed data is again very similar to the previous examples. Note that in this example the parameter priors are specified explicitly (although those given are the same as the defaults). The `+1` in the hyperparameter specification is because a constant term is included in the additive formulation for each node.

```
> mydag<-matrix( 0, 33, 33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## create empty DAGs
> banned.mix<-matrix( 0, 33, 33);
> colnames(banned.mix)<-rownames(banned.mix)<-names(var33);
> retain.mix<-matrix( 0, 33, 33);
> colnames(retain.mix)<-rownames(retain.mix)<-names(var33);
> ## setup distribution list for mixed node
> mydists.mix<-list( v1 = "binomial", v2 = "gaussian",
+        v3 = "binomial", v4 = "binomial", v5 = "gaussian",
+        v6 = "binomial", v7 = "gaussian", v8 = "gaussian",
+        v9 = "binomial", v10 = "binomial", v11 = "binomial",
+        v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
+        v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
+        v18 = "binomial", v19 = "binomial", v20 = "binomial",
+        v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
+        v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
+        v27 = "binomial", v28 = "binomial", v29 = "gaussian",
+        v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
+        v33 = "gaussian");
> ## build cache of all the local computations
> ## this information is needed later when running a model search
> mycache.mix<-buildscorecache( data.df=var33,
+              data.dists=mydists.mix, dag.banned=banned.mix,
+              dag.retained=retain.mix, max.parents=1);
> # Run a single search heuristic for an additive BN
> heur.res.mix<-search.hillclimber( score.cache=mycache.mix,
+               num.searches=1, seed=0, verbose=FALSE,
+               trace=FALSE, timing.on=FALSE);
> # Setting trace=TRUE, the majority consensus network is
> # plotted as the searches progress
```

5. Multiple Search Strategies

To estimate a robust additive BN for a given dataset is it necessary to run many searches and then summarize the results of these searches. The function `search.hillclimber()` with `num.searches>1` run multiple searches. It is necessary to use a single joint node cache over all searches, using the function `buildscorecache`.

Conceptually it may seem more efficient to use one global node cache to allow node information to be shared between different searches, however, in practice as the search space is so vast for some problems this can result in extremely *slow* searches. As the cache becomes larger it can take much more time to search it (and it may need to be searched a very large number of times) than to simply perform the appropriate numerical computation. Profiling using the google performance tool google-pprof suggests that more than 80% of the computation time may be taken up by lookups. When starting searches from different random places in the model space the number of individual node structures in common between any two searches, relative to the total number of different node structures searched over can be very small meaning a common node cache is inefficient. This may not be the case when starting networks are relatively similar.

To help with performance monitoring it is possible to turn on timings using `timing.on=TRUE` which then outputs the number of seconds of CPU time each individual search takes (using standard libc functions declared in time.h).

```
> mydag<-matrix( 0, 33, 33);
> colnames(mydag)<-rownames(mydag)<-names(var33);#set names
> ## create empty DAGs
> banned.mix<-matrix( 0, 33, 33);
> colnames(banned.mix)<-rownames(banned.mix)<-names(var33);
> retain.mix<-matrix( 0, 33, 33);
> colnames(retain.mix)<-rownames(retain.mix)<-names(var33);
> ## setup distribution list for mixed node
> mydists.mix<-list( v1 = "binomial", v2 = "gaussian",
+          v3 = "binomial", v4 = "binomial", v5 = "gaussian",
+          v6 = "binomial", v7 = "gaussian", v8 = "gaussian",
+          v9 = "binomial", v10 = "binomial", v11 = "binomial",
+          v12 = "binomial", v13 = "gaussian", v14 = "gaussian",
+          v15 = "binomial", v16 = "gaussian", v17 = "gaussian",
+          v18 = "binomial", v19 = "binomial", v20 = "binomial",
+          v21 = "binomial", v22 = "gaussian", v23 = "gaussian",
+          v24 = "gaussian", v25 = "gaussian", v26 = "binomial",
+          v27 = "binomial", v28 = "binomial", v29 = "gaussian",
+          v30 = "gaussian", v31 = "gaussian", v32 = "binomial",
+          v33 = "gaussian");
```

```
> n.searches<- 10; # example only - must be much larger in practice
> ## parent limits
> max.par<-1 #only 1 because take some minutes for buildscorecache()
> ## now build cache
> mycache.mix<-buildscorecache(data.df=var33, data.dists=mydists.mix,
+ dag.banned=banned.mix, dag.retained=retain.mix, max.parents=max.par)
> # repeat but this time have the majority consensus network plotted
> # as the searches progress
> myres.mlp<-search.hillclimber(score.cache=mycache.mix,
+             num.searches=n.searches, seed=0, verbose=FALSE,
+             trace=FALSE, timing.on=FALSE);
```

5.1. Creating a Summary Network - Majority Consensus

Having run many heuristic searches, then the next challenge is to summarise these results to allow for ready identification of the joint dependencies most supported by the data. One common, and very simple approach is to produce a single robust BN model of the data mimicing the approach used in phylogenetics to create majority consensus trees. A majority consensus DAG is constructed from all the arcs present in at least 50% of the locally optimal DAGs found in the search heuristics. This creates a single summary network. Combining results from different runs of `search.hillclimber()` or `search.hillclimber()` is straightforward, although note that it is necessary to check for duplicate random starting networks, as while highly unlikely this is theoretically possible. The following code provides a simple way to produce a majority consensus network and Figure 5 shows the resulting network - note that this is an example only and many thousands of searches may need to be conducted to achieve robust results. One simple ad-hoc method for assessing how many searches are needed is to run a number of searches and split the results into two (random) groups, and calculate the majority consensus network within each group. If these are the same then it suggests that sufficient searches have been run. To plot the majority consensus network use the result of the function `search.hillclimber`, see below for some example.

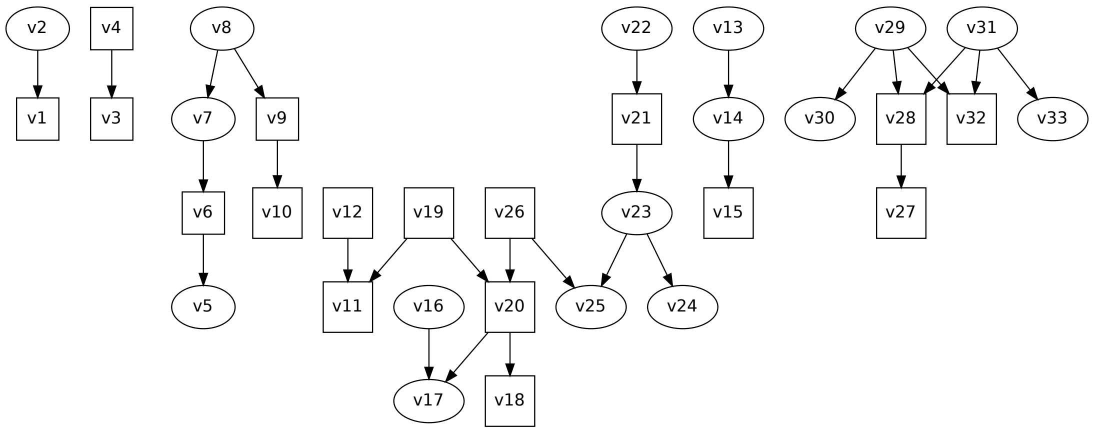

Figure 5: Example majority consensus network (from the results of only 10 searches)

```
> tographviz(dag= myres.mlp$consensus, data.df=var33,
+ data.dists=mydists.mix, outfile="dagcon.dot");#create file
> # dagcon.dot can then be processed with graphviz
> # unix shell "dot -Tpdf dagcon.dot -o dagcon.pdf" or use
> # gedit if on Windows
```

6. Creating a Summary Network - Pruning

Rather than use the majority consensus network as the most appropriate model of the data, an alternative approach is to choose the single best model found during a large number of searches. To determine sufficient heuristic searches have been run to provide reasonable coverage of all the features of the model landscape, then again checking for a stable majority consensus network as in Section 5.1, seems a sensible approach. Once the best overall DAG has been identified then the next task is to check this model for over-fitting. Unlike with the majority consensus network, which effective "averages" over many different competing models and therefore should generally comprise only robust structural features, choosing the DAG from a single model search is far more likely to contain some spurious features. When dealing with smaller data sets, say, of several hundred observations then this is extremely likely, as can easily be demonstrated using simulated data. A simple assessment of overfitting can be made by comparing the number of arcs in the majority consensus network with the number of arcs in the best fitting model. We have found that in larger data sets the majority consensus and best fitting model can be almost identical, while in smaller data sets the best fitting models may have many more arcs - suggesting a degree of overfitting.

An advantage of choosing a DAG from an individual search is that unlike averaging over lots of different structures, as in the construction of a majority consensus network, the model chosen here has a structure which was actually found during a search across the model landscape. In contrast, the majority consensus network is a derived model which may never have been found chosen during even an exhaustive search, indeed it may even comprise of contradictory features as is a usual risk in averaging over different explanations (models) of data. In addition, a majority consensus network need also not be acyclic, although in practice this can be easily corrected by reversing one or more arcs to produce an appropriate DAG.

A simple compromise between the risk of over-fitting in choosing the single highest scoring DAG, and the risk of inappropriately averaging across different distinct data generating processes, is to prune the highest scoring DAG using the majority consensus model. In short, an element by element multiply of the highest scoring DAG and the majority consensus DAG, which gives a new DAG which only contains the structural features in *both* models.

7. Creating a Summary Network - Parametric Bootstrapping

In (Friedman, Goldszmidt, and Wyner 1999) a general approach for using parametric bootstrapping to select BN models/DAG structures was presented. Such approaches can be reasonably easily implemented by using readily available Markov chain Monte Carlo sampling software such as JAGS or WinBUGS. The basic idea is to take the structure with the best network score and then code it up in either JAGS or WinBUGS, and use these samplers to generate bootstrap data sets from this model. That is, independent realisations from the model which can be used to generate a data set of the same size as the observed data. Given this bootstrap data, then the BN model search is repeated treating this as the observed data. By generating many bootstrap data sets and conducting searches on each, then this allows us to estimate the percentage support for each arc in the highest scoring model. Another way to put this is that we find out how many of the arcs in the highest scoring model can be "recovered" from a data set of the size as that actually observed. Obviously, the more data, the more statistical power, and the more structural features which can be recovered. For arcs with a lower level of support, e.g. <50%, then these can be pruned from the best fitting model, the assumption being that these are potentially as a result of overfitting. The resulting model - possibly with arcs pruned off from the original model - is then our chosen model of the data.

Using a 50% threshold for arc support is intuitively reasonable as can be seen by considering a model of a single node. Suppose there are two covariates, and the "response" variable (the node) is almost deterministically dependent with each of these variables (that is overwhelming statistical support), and that the two covariates are almost exactly collinear. In which case only one arc will be needed in the model, and running random restart heuristic searches in this case will result in approximately 50% of search results suggesting an arc from covariate one to the response variable, and the other 50% for an arc from covariate two to the response variable. Therefore, despite the fact that each of these variables is almost surely (with probability=1) dependent with the response variable each arc cannot exceed 50% support. This is an idealized example but provides an intuitive argument as to why 50% is a reasonable threshold above which we can be fairly confident that the arcs may be a real, rather than spurious statistical feature.

While parametric bootstrapping is a general technique is well established and conceptually elegant, this may in practice not be computationally feasible. Even if taking the least demanding approach of conducting only one heuristic search per bootstrap data set, the number of data sets/searches required in order to get robust % support values for each arc in the best fitting model may be large, and beyond what is reasonably possible even using high performance computing (HPC) hardware. We would suggest that it is at least good practice to investigate the feasibility of this approach. To that end we next outline how to perform parametric bootstrapping using **abn** and `JAGS` and all the relevant source files are included in the `abn/bootstrapping_example` library subdirectory.

7.1. Steps Necessary to Perform Parametric Bootstrapping

Given a BN model - DAG structure plus parameter priors - then the first step is to estimate the posterior parameters. The second step is to implement the DAG structure together with the posterior parameters into the language used by `JAGS`, which is very similar to that used in `WinBUGS`. Generally speaking, the posterior parameters in an additive BN model need not conform to any standard probability distribution as these are non-conjugate models. To make the implementation as general as possible, rather than, for example, attempting to match each posterior distribution to, say, the closest shaped Gaussian density, we present an approach which allows all posterior parameters to be from a non-standard - bespoke - distribution. This is implemented in `JAGS` by discretising each posterior density across a fine grid and using a discrete sampler, `dcat()` in `JAGS`.

Estimating Posterior Densities

The function `fitabn()` with `compute.fixed=TRUE` uses Laplace approximations to estimate the posterior density of each parameter in a BN model. An appropriate domain (range) for each parameter *must* be supplied by the user which is done by some trial and error to find where about on the real line the density resides - it will be close to the origin either through the use of a logistic link function or through the standardisation of the Guassian node. An initial guess of (-2,2) is often a good starting point. It is crucially important that a sufficiently wide range is given so that “all” of the upper and lower tails of the distribution are included, e.g. the range should be where an R density plot first drops to approximately zero at each tail. Note that `fitabn()` works with one node and one parameter in that node at a time. It is not necessary - but does no harm - to specify the full DAG, but all that is needed is the node and its parents.

We now follow the example contained in the `abn/bootstrapping_example` library subdirectory. We use a second data set included with **abn**, called `pigs.1par` which is again a synthetic dataset generated from analyses of real data. The first step is define the model of interest, `mydag`, and then estimate the posterior densities. The posterior parameters calculated are the marginal effects, that is all other model parameters (at the given node) are integrated out. At each node - which is defined using argument `marginal.node` - the intercept term can be estimated using `marginal.param=1` (see below) and for Gaussian nodes the precision parameter can be estimated using `prec`. The remaining parameters are the mean covariate effects, for example using `marginal.node=1` and `marginal.param=2` in `fitabn()` gives the posterior marginal density for covariate `D2` for the response variable `D1`. Note that by default all Gaussian variables are standardised to a mean of zero and a standard deviation of one.

```
> #specific a DAG model - the model we wish to use to perform
> #parametric bootstrapping
> mydag.pigs<-matrix(c(
+ # D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 Year Loc.x Loc.y
+   0, 1, 0, 0, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # D1
+   0, 0, 1, 0, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # D2
+   0, 0, 0, 1, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # D3
+   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   0,    1,     0,     # D4
+   0, 0, 0, 0, 0, 1, 0, 0, 0, 0,   0,    0,     0,     # D5
+   0, 0, 0, 1, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # D6
+   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   1,    0,     0,     # D7
+   0, 0, 0, 0, 0, 0, 0, 0, 0, 1,   0,    0,     0,     # D8
+   0, 1, 0, 0, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # D9
+   0, 0, 0, 0, 0, 0, 0, 0, 1, 0,   0,    0,     0,     # D10
+   0, 0, 0, 0, 0, 0, 0, 0, 0, 0,   0,    0,     0,     # Year
+   0, 0, 0, 0, 0, 0, 1, 0, 0, 0,   0,    0,     0,     # Loc.x
+   0, 0, 0, 0, 0, 0, 1, 0, 0, 0,   0,    0,     0      # Loc.y
+                  ),byrow=TRUE,ncol=13);
> colnames(mydag.pigs)<-rownames(mydag.pigs)<-names(pigs.1par);
> ## setup distribution list for each "pigs" node
> mydists.pigs <- list( D1 = "binomial", D2 = "binomial",
+         D3 = "binomial", D4 = "binomial", D5 = "binomial",
+         D6 = "binomial", D7 = "binomial", D8 = "binomial",
+         D9 = "binomial", D10 = "binomial", Year = "gaussian",
+         Loc.x = "gaussian", Loc.y = "gaussian");
> # Node D1|D2 e.g. logit(P(D1=TRUE)=constant+coeff*D2
> # first get the posterior density for the constant and
> # get grid of discrete values x and f(x)
> marg.D1.1<-fitabn( data.df=pigs.1par, dag.m=mydag.pigs,
+              data.dists=mydists.pigs, compute.fixed=TRUE,
+            marginal.node=1, marginal.param=1, # intercept
+            variate.vec=seq(from=1,to=1.7,len=1000),
+            verbose=FALSE, n.grid=1000);
> print(names(marg.D1.1$marginals));
```

```
[1] "D1|(Intercept)"
```

```
> # now repeat for the slope term coeff
> marg.D1.2<-fitabn( data.df=pigs.1par, dag.m=mydag.pigs,
+            data.dists=mydists.pigs, compute.fixed=TRUE,
+            marginal.node=1, marginal.param=2, # slope
+            variate.vec=seq(from=0.6,to=1.5,len=1000),
+            verbose=FALSE, n.grid=1000);
> print(names(marg.D1.2$marginals));
```

```
[1] "D1|D2"
```

The parameters can be estimated one at a time by manually giving a grid. We plot the marginal posterior densities for node D1 with one parent, the covariate D2, as in the following model:

$$\text{logit}\{P(\text{D3}=1)\} = \beta_{D1,0} + \beta_{D1,1} \cdot \text{D2}$$

Figure 6 shows an example of posterior densities estimated using `fitabn()`, all posterior densities for all parameters in the additive BN can be estimated in the same way.

The file `calculate_marginalDensities.R` in `abn/bootstrapping_example` contains R code for estimating all the marginal parameters in the DAG `mydag` given above. This file is documented. It calculates all the marginal distributions and then writes them out to a file `post_params.R` which is in the R dump format which can be read into JAGS.

The remaining files are `script1.R` which is the script which runs the `JAGS` MCMC sampling. The `JAGS` package is open source and can be downloaded from sourceforge along with appropriate documentation. The file `simulate_1par.bug` contains the DAG implemented into the `JAGS` language along with the posterior parameter estimates, the creation of this file is the main task involved in the parametric bootstrapping. Once the `script1.R` has completed (this script is run simply by typing `jags script1.R` at the command line), then the bootstrap data set generated is contained in the file `outchain1.txt` and `outchain2.txt`. Running two (or more) chains allows for checking of convergence. The script is set up to generate 10000 realisations, to get a single bootstrap data set we would trim this down to 9011 observations which is that same size as the original data `pigs.1par`. Once we have this bootstrap data set then we simply run a model search on this, for example using `search.hillclimber()` as detailed above.

To automate the parametric bootstrapping one option is to edit the `script1.R` so that sufficient realisations are generated to create many independent bootstrap data sets, for example generate 1000 $\times$ 9011 realisations.
For a better overview of the automatization procedure, refer to the website www.r-bayesian-networks.org, in particular see the section "Case Studies/ Case Study One/ Automating for use on a cluster".

```
> par( mar=c(8.8,8.2,3.1,3.1),mgp=c(4,2,0));
> par( cex.axis=2,cex.lab=2,cex.main=2);
> par( las=1,xaxs="i",yaxs="i");
> plot( marg.D1.1$marginals[["D1|(Intercept)"]],xlab="Log odds",
+       ylab="Density",type="l",axes=!FALSE,xlim=c(0,2),
+       ylim=c(0,7),col="brown",lwd=3,lty=1);
> lines( marg.D1.2$marginals[["D1|D2"]],
+       col="blue",lwd=3,lty=6);
> legend( "topleft",legend=c(expression(paste("f(",beta[D1*","~0],
+             "|D)= intercept node D1",sep="")),
+         expression(paste("f(",beta[D1*","~1],
+             "|D)= effect of D2 at node D1",sep=""))), cex=1.5,
+         col=c("brown","blue"),lty=c(1,6),lwd=5, bty='n');
```

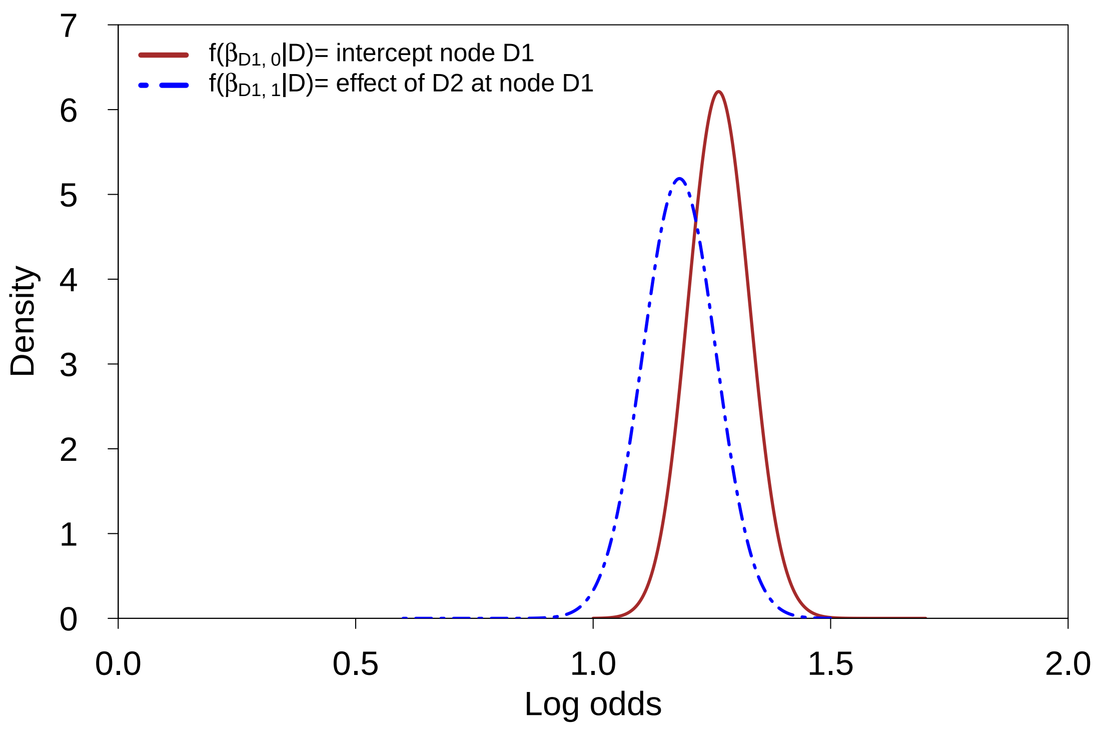

Figure 6: Posterior densities for intercept and slope in node D1 in pigs.1par

8. Order Based Searches

It is generally not feasible to iterate over all possible DAG structures when dealing with more than a handful of variables, hence the reliance on heuristic searches. It is also extremely difficult to construct efficient Monte Carlo Markov chain samplers across BN structures. A solution to this was proposed in (Friedman and Koller 2003) where rather than sample across DAGs, it was proposed to sample across node orderings. A node order is simply the set of integers 1 through n, where n is the number of variables in the data. A DAG is consistent with an ordering if for each node in the order its parents come before it. For example a DAG with only an arc from 1→2 is consistent with ordering $1, 2, 3, 4$ as the parent 1 comes before 2, but a DAG with an arc from 2→1 is not consistent with this ordering. In effect, each order is a collection of DAGs, and note that each DAG may be consistent with multiple orders, i.e. the empty DAG is consistent with every possible ordering. This introduces bias, in that averaging across orders need not give the same results as averaging across DAGs, if the latter were possible. This is relevant when estimating posterior probabilities of individual structural features, and is baised towards more parsimonious features as they are consistent with more orders. Note that this bias does not apply to maximising across orders, as in finding most probable structures (see later). The big advantage of searching across orders is that there are $n!$ different orders compared to a reasonably tight upper bound of $2^{\binom{n}{2}}$ for different DAGs.

There are (at least) two approaches for searching across orders. The first is to construct a Markov chain which samples from the posterior distribution of all orders, and is the approach presented in (Friedman and Koller 2003). Alternatively, in (Koivisto and Sood 2004) an exact method is proposed which rather than sample across orders, performs an exhaustive search. This has the advantage that it can also be used to find the globally optimal DAG of the data - the most probable structure - as well as posterior probabilities for structural features, such as individual arcs. The drawback is that this exact approach is only feasible with smaller number of variables e.g. up to 12 or 13 when dealing with additive models. For the code provided in **abn** this exact approach is readily feasible up to 20 variables using typical desktop computing, and potentially up to 25 variable with access to a shared memory cluster computer.

8.1. Most Probable Structure

Using the exact order based method due to (Koivisto and Sood 2004) it is also possible to identify the DAG with globally best network score. Identification of a most probable structure is split into two parts. Firstly we calculate a cache of individual node scores, for example using `buildscorecache()`. Next, an exhaustive order based structural search is carried out using the function `mostprobable` which relies on the information in the node cache.

As in the heuristic searches it is possible to ban or retain certain arcs, for example when splitting multinomial variables. This is done in the node cache computation step. There are two different structural priors implemented in this search, the first in the uniform prior where all structures (all parent combinations) are equally likely. This is the default `prior.choice=1` in `mostprobable` and the other functions. Also implemented is the prior used in (Koivisto and Sood 2004) where all parent combinations of equal cardinality are equally weighted, this is `prior.choice=2`. The latter does give the same prior weight to a parent combination with no parents and a parent combination comprising off all possible parents (since there is only one choice of each, n-1 choose 0 and n-1 choose n-1). This may not be desirable but is included as a prior for completness. Note that the order based search is exact in the sense that it will identify a DAG who score is equal to the best possible score if it was possible to exhaustive search across all DAGs. For example, if using `prior.choice=1` then the network found should have a score greater than or equal to that found using the previously described heuristic searches. The structure found need not be unique in that others may exist with the same globally optimal score, the order based search is only guaranteed to find one such structure.

To calculate the most probable structure we again use `buildscorecache()` to calculate a cache of individual node scores. Next, the function `mostprobable()` does the actual exhaustive order based search, and works for both conjugate and additive models since as with calculating the posterior probabilities this step only involves structural searching and is not concerned with the precise parameterisation of each BN model.

Below are some examples of how to find the most probable structure - these are very small examples, but work identical for larger data sets, but with considerably increased computational time. The low memory versions give copius output so its possible to see what is happening during execution - these are only designed for use of larger problems e.g. 20+ variables.

```
> pigs<-pigs.1par[,c(1:8,12,13)];
> # all 9011 observation but limit to 10 variables
> # using all 13 variables in pigs will take several
> # hours of cpu time
> max.par <- 1
> # setup the distribution for each "pigs subset nodes
> mydists.pigs.sub <- list( D1 = "binomial", D2="binomial",
+          D3 = "binomial", D4 = "binomial", D5 = "binomial",
+          D6 = "binomial", D7 = "binomial", D8 = "binomial",
+          Loc.x = "gaussian", Loc.y = "gaussian");
> banned.pigs.sub<-matrix( 0, 10, 10);#banlist with no constraints
> colnames(banned.pigs.sub)<-rownames(banned.pigs.sub)<-names(pigs);
> retain.pigs.sub<-matrix( 0, 10, 10);#retainlist without constraints
> colnames(retain.pigs.sub)<-rownames(retain.pigs.sub)<-names(pigs);
> #compute node cache - note restriction of max. 1 parent
> # per node, this should be increased as necessary
> system.time( mynodes.add<-buildscorecache( data.df=pigs,
+         data.dists=mydists.pigs.sub, max.parents=max.par,
+         dag.banned=banned.pigs.sub, dag.retained=retain.pigs.sub));
```

```
   user  system elapsed
  0.583   0.004   0.589
```

```
> # now find the globally best model using previous node cache - so
> # we are only looking for the best DAG, most probable network,
> # within the scope of no more than one parent per node
> map.1par.10var<-mostprobable( score.cache=mynodes.add);
```

```
Step1. completed max alpha_i(S) for all i and S
Total sets g(S) to be evaluated over: 1024
```

```
> tographviz(dag=map.1par.10var,data.df=pigs,
+            data.dists=mydists.pigs.sub,
+            outfile="map1_10var.dot");#create file
> # map1_10var.dot can then be processed with graphviz
> # unix shell "dot -Tpdf map1_10var.dot -o map1_10var.pdf" or
> # use gedit if on Windows
```

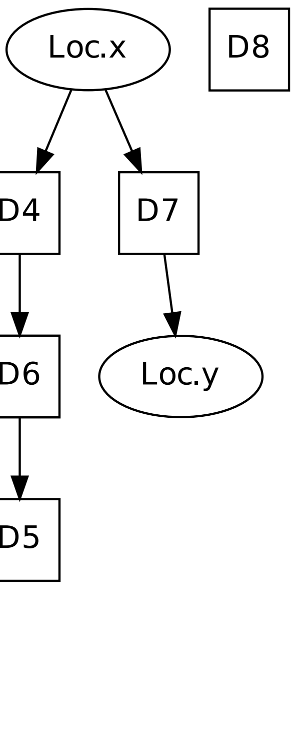

Figure 7: Most probable DAG for data in pigs.1par with variables D1-D8, Loc.x and Loc.y, and after imposing a limit of no more than one parent per node

Figure 7 shows a most probable DAG using a subset of variables from pigs.1par and a parent limit of one per node.

```
> pigs.all<-pigs.1par;#all observations all variables
> ## setup distribution list for each node
> mydists.pigs <- list( D1 = "binomial", D2 = "binomial",
+         D3 = "binomial", D4 = "binomial", D5 = "binomial",
+         D6 = "binomial", D7 = "binomial", D8 = "binomial",
+         D9 = "binomial", D10 = "binomial", Year = "gaussian",
+         Loc.x = "gaussian", Loc.y = "gaussian");
> banned.pigs <- matrix( 0, 13, 13);
> colnames(banned.pigs)<-rownames(banned.pigs)<-names(pigs.all);
> retain.pigs <-  matrix( 0, 13, 13);
> colnames(retain.pigs)<-rownames(retain.pigs)<-names(pigs.all);
> system.time( mynodes.add.all<-buildscorecache( data.df=pigs.all,
+         max.parents=1, data.dists=mydists.pigs,
+         dag.banned=banned.pigs, dag.retained=retain.pigs));


   user  system elapsed
  0.977   0.003   0.981

> ## now for most probable network of all DAGs where each node has
> ## at most one arc
> system.time( map.1par<-mostprobable( score.cache=mynodes.add.all,
+                                      prior.choice=1));


Step1. completed max alpha_i(S) for all i and S
Total sets g(S) to be evaluated over: 8192
   user  system elapsed
  0.231   0.001   0.232

> tographviz( dag=map.1par,data.df=pigs.all, data.dists=mydists.pigs,
+            outfile="map_1par.dot");#create file
> # mydag.dot can then be processed with graphviz
> # unix shell "dot -Tpdf map_1par.dot -o map_1par.pdf" or use gedit
> # if on Windows
```

Figure 8 shows the most probable DAG considering all variables in pigs.1par and a parent limit of one per node.

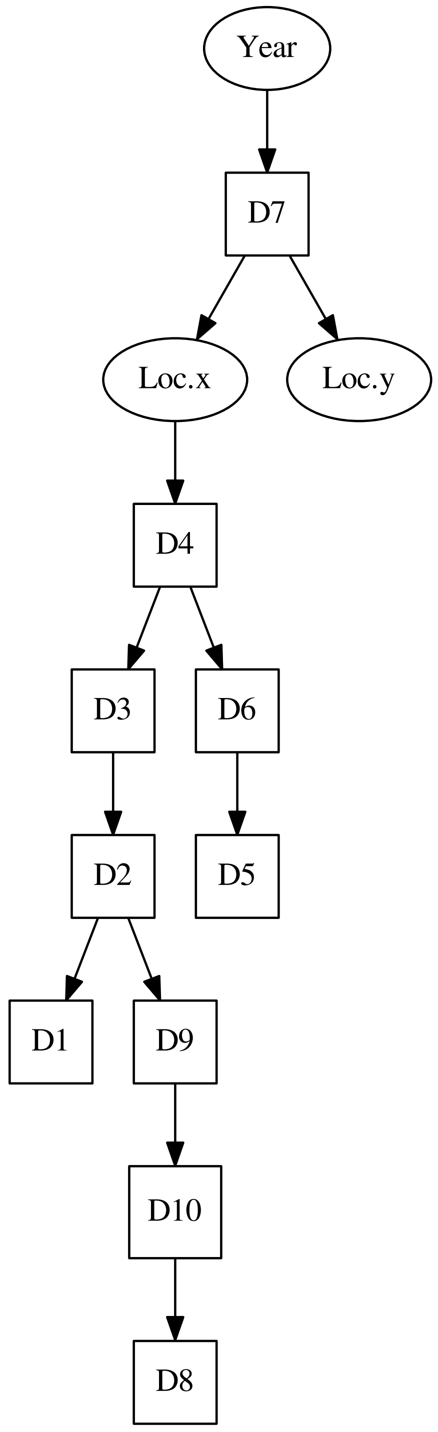

Figure 8: Most probable DAG for data in pigs.1par imposing a limit of no more than one parent per node

9. Summary

The **abn** library provides a range of Bayesian network models to assist with identifying statistical dependencies in complex data, in particular models which are multidimensional analogues of generalised linear models. This process is typically referred to as structure learning, or structure discovery, and is computational extremely challenging. Heuristics are the only options for data comprising of larger numbers of variables. As with all model selection, over-modelling is an everpresent danger and using either: i) summary models comprising of structural features present in many locally optimal models or else; ii) using parametric bootstrapping to determine the robustness of the features in a single locally optimal model are likely essential to provide robust results. An alternative presented was exact order based searches, in particular finding the globally most probable structure. This approach is appealing as it is exact, but despite collapsing DAGs into orderings for larger scale problems it may not be feasible. For further in-depth analysis about **abn** refer to the website: www.r-bayesian-networks.org.

References

Buntine W (1991). "Theory refinement on Bayesian networks." In *Proc. Seventh Conference on Uncertainty in Artificial Intelligence, pp. 52-60. Morgan Kaufmann, Los Angeles, CA, USA*.

Friedman N, Goldszmidt M, Wyner A (1999). "Data analysis with Bayesian networks: A Bootstrap approach." In *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99) (pp.206-215). San Francisco: Morgan Kaufmann*.

Friedman N, Koller D (2003). "Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks." *Machine Learning*, **50**(1-2), 95–125.

Heckerman D, Geiger D, Chickering DM (1995). "Learning Bayesian Networks - The Combination of Knowledge And Statistical Data." *Machine Learning*, **20**(3), 197–243.

Hodges AP, Dai DJ, Xiang ZS, Woolf P, Xi CW, He YQ (2010). "Bayesian Network Expansion Identifies New ROS and Biofilm Regulators." *Plos One*, **5**(3), e9513.

Jensen FV (2001). *Bayesian Network and Decision Graphs*. Springer-Verlag, New York.

Koivisto M, Sood K (2004). "Exact Bayesian structure discovery in Bayesian networks." *Journal of Machine Learning Research*, **5**, 549–573.

Lauritzen SL (1996). *Graphical Models*. Oxford Univ Press, New York.

Lewis FI, Brulisauer F, Gunn GJ (2011). "Structure discovery in Bayesian networks: An analytical tool for analysing complex animal health data." *Preventive Veterinary Medicine*, **100**(2), 109–115.

Needham CJ, Bradford JR, Bulpitt AJ, Westhead DR (2007). "A primer on learning in Bayesian networks for computational biology." *Plos Computational Biology*, **3**(8), e129.

Poon AFY, Lewis FI, Pond SLK, Frost SDW (2007). "Evolutionary interactions between N-linked glycosylation sites in the HIV-1 envelope." *Plos Computational Biology*, **3**(1), 110–119.

Rijmen F (2008). "Bayesian networks with a logistic regression model for the conditional probabilities." *International Journal of Approximate Reasoning*, **48**(2), 659–666.

Affiliation:

Fraser Ian Lewis
Office of Health Economics
United Kingdom, London
E-mail: flewis@ohe.org

Marta Pittavino
Institute of Mathematics, University of Zurich
Winterthurerstrasse 270, Zurich 8057, Switzerland
E-mail: marta.pittavino@math.uzh.ch

Reinhard Furrer
Institute of Mathematics, University of Zurich
Winterthurerstrasse 270, Zurich 8057, Switzerland
E-mail: reinhard.furrer@math.uzh.ch