







## Additive Bayesian Network Modeling with the R Package `abn`

Gilles Kratzer   
University of Zurich

Fraser Lewis   
GSK Pharmaceutical  
Manufacturing

Arianna Comin   
Swedish National  
Veterinary Institute

Marta Pittavino   
University of Geneva

Reinhard Furrer   
University of Zurich

---

### Abstract

The R package `abn` is designed to fit additive Bayesian network models to observational datasets and contains routines to score Bayesian networks based on Bayesian or information theoretic formulations of generalized linear models. It is equipped with exact search and greedy search algorithms to select the best network, and supports continuous, discrete and count data in the same model and input of prior knowledge at a structural level. The Bayesian implementation supports random effects to control for one-layer clustering. In this paper, we give an overview of the methodology and illustrate the package's functionality using a veterinary dataset concerned with respiratory diseases in commercial swine production.

*Keywords:* structure learning, graphical models, greedy search, exact search, scoring algorithm, GLM, graph theory.

---

## 1. Introduction

Bayesian network (BN) modeling has an impressive track record in the domain of systems epidemiology (McCormick 2014; Hartnack *et al.* 2019), especially in veterinary epidemiology (Ruchti, Kratzer, Furrer, Hartnack, Würbel, and Gebhardt-Henrich 2019; Comin, Jeremiasson, Kratzer, and Keeling 2019). It is particularly well-suited for gaining a better understanding of the underlying structure of data when scientific knowledge is at an early stage. Additive Bayesian networks (ABN) is class of Bayesian networks where the relationships are all assumed to be additive and is a rich multivariate extension of classical additive modeling

approaches, such as least squares regression or logistic regression, which are ubiquitous in epidemiology. ABN models seek to discriminate directly from indirectly related variables, with a key use case being hypotheses generation from messy or complex observational data. In contrast to techniques such as structural equation modeling (Hair, Anderson, Tatham, and Black 1998) ABN is a data-driven approach (Lewis and Ward 2013) and so does not require expert knowledge in order to determine an optimal model structure, but such knowledge can be incorporated if desired. In contrast to other BN modeling approaches, ABNs offer additional flexibility as they can comprise of variables from different types of distributions, for example count, categorical and continuous variables can all be mutually dependent in an ABN model. In addition, ABNs also support adjustment for some types of clustering/correlation of observations, and each network can be readily scored for goodness of fit and effect sizes estimation. This paper presents an R implementation of the ABN methodology with case study data used to illustrate the possible range of analyses.

R is an open-source, reliable and easy-to-use environment for statistical computing (R Core Team 2022). It is very popular in the epidemiological community. In this paper, ABN refers to the methodology and **abn** refers to the R package.

The aim of the R package **abn** (Furrer, Kratzer, and Lewis 2023) is to provide researchers with a free implementation of a set of functions to score, select, analyze and report an ABN based analysis. The package includes Bayesian and information theoretic scoring, exact and greedy search algorithm functionality. The R package **abn** is also equipped with ancillary functions to simulate and manipulate ABN models. The aim of the R package **abn** is to provide researchers with a free implementation of a set of functions to score, select, analyze and report an ABN based analysis. The package includes Bayesian and information theoretic scoring, exact and greedy search algorithm functionality. The R package **abn** is also equipped with ancillary functions to simulate and manipulate ABN models. The package is available through the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=abn> and at <https://git.math.uzh.ch/reinhard.furrer/abn>, the later containing the devel versions and issue tracker.

A unique feature of the R package **abn** compared to other network modeling R packages is the Bayesian-based scoring system, which makes the ABN methodology theoretically sound and computationally competitive thanks to the internal use of iteratively nested Laplace approximations (INLA, Rue, Martino, and Chopin 2009, available at <https://www.R-INLA.org/download-install>). From an applied perspective, a regression framework is suitable for analyses that target data modeling and seek to report insights from observational data. A byproduct of the Bayesian regression framework used by the R package **abn** is the ability to adjust an analysis for clustering, which is a common requirement in systems epidemiology and not readily possible in other network modeling R packages. Beyond its appealing theoretical framework, the R package **abn** is equipped with the only R implementation of an exact search based on dynamical programming for BN structure discovery (Koivisto and Sood 2004).

The structure of this paper is as follows: first, we finish this section with a short motivating example and alternative approaches for modeling BNs. Then, Section 2 comprehensively describes the ABN methodology. Section 3 lists and describes the functionality of the R package. It includes simulation studies which compare the efficiency of the various scores implemented. Section 4 presents a case study using data from the field of veterinary epidemiology. We conclude and summarize the article in Section 6. The appendix provides more technical details.

Variable	Meaning	Distribution
Asia	Recent visit to Asia	Binomial
Smoking	Actual smoking status	Binomial
Tuberculosis	Tuberculosis status	Binomial
LungCancer	Lung cancer status	Binomial
Bronchitis	Bronchitis status	Binomial
Either	Tuberculosis versus lung cancer/bronchitis	Binomial
XRay	Chest X-ray	Binomial
Dyspnea	Shortness-of-breath status	Binomial

Table 1: Description of the variables provided by the dataset `asia` from `bnlearn`. All variables are encoded with levels “yes” and “no”.

### 1.1. Simple introductory example

We start illustrating the main functionality of the R package `abn` by analyzing the so-called `asia` dataset provided by the R package `bnlearn` (Scutari 2010). The `asia` dataset is a small synthetic dataset from Lauritzen and Spiegelhalter (1988) about lung diseases (tuberculosis, lung cancer and bronchitis) and visits to Asia. In total, the dataset consists of eight binary variables with 5000 observations, see Table 1. As this dataset has been used in various illustrations, we do not present it in detail here, nor do we perform an initial exploratory data analysis. The R package `abn` and its dependencies are available from CRAN except for `INLA` which is available at <https://www.R-INLA.org/download-install> and `Rgraphviz` (Hansen *et al.* 2022) from `Bioconductor`. Additionally, the package requires the `JAGS` computing library (Plummer 2003) and the GNU Scientific Library (`GSL` version  $\geq 1.12$ , <https://www.gnu.org/software/gsl>).

We start with loading the data. The advantage of using the R package `abn` over a classical approach is that we take all data into account without making an a priori choice of what are the independent and the dependent variables.

```
R> library("abn")
R> data("asia", package = "bnlearn")
R> colnames(asia) <- c("Asia", "Smoking", "Tuberculosis", "LungCancer",
+   "Bronchitis", "Either", "XRay", "Dyspnea")
```

We now determine the relationships between the data. A more theoretical and detailed explanation is given in Sections 2 and 3.

```
R> distrib <- as.list(rep("binomial", 8))
R> names(distrib) <- names(asia)
R> mycache <- buildScoreCache(data.df = asia, data.dist = distrib,
+   max.parents = 4)
R> mp.dag <- mostProbable(score.cache = mycache)
```

We have used an ABN model to estimate all the relationships in the available data, this is typically referred to as structure learning in the BN literature. Once we have estimated

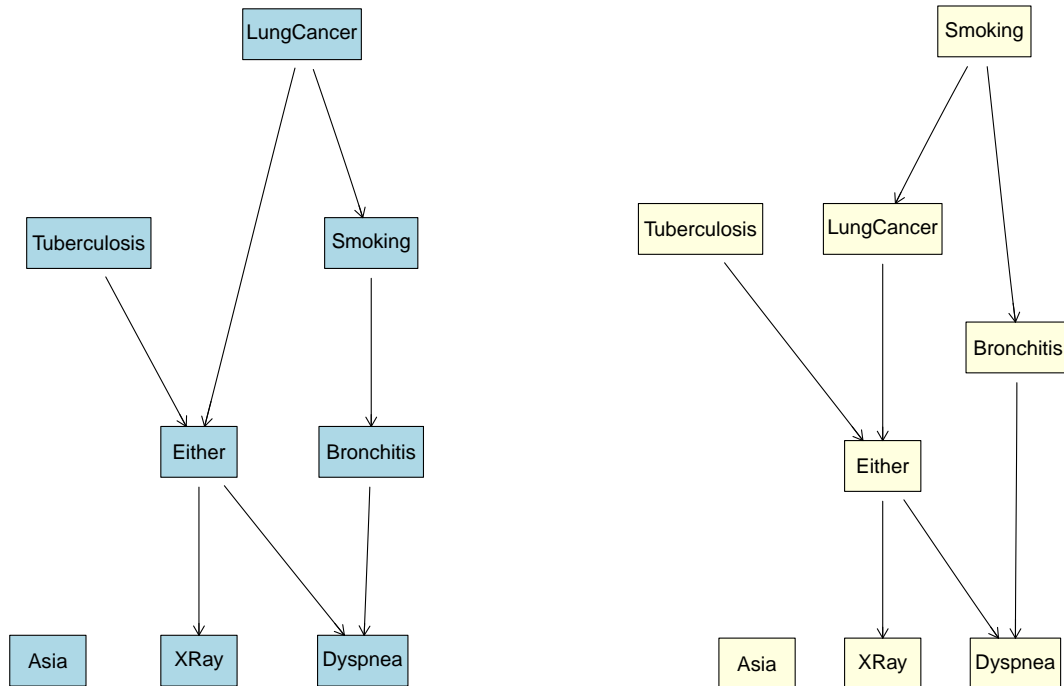


Figure 1: Network for `asia` dataset. Left: unconstrained, right: constrained. The constraint imposes a link from the node `Smoking` to the node `LungCancer`.

the model structure we will estimate the parameters in this model, e.g., odds ratios of the conditional effects. This step is referred to as parameter learning.

```
R> fabn <- fitAbn(object = mp.dag)
R> plot(fabn, fitted.values = NULL)
```

The resulting structure is visualized in the left panel of Figure 1. This example highlights that the R package `abn` does not model causality. It is possible, however, to structurally constrain the modeling procedure to retain or ban arcs between variables. Here, we impose an arrow from the node `Smoking` to `LungCancer` with `dag.retained = ~ LungCancer|Smoking` (we discuss alternative ways to impose structural constraints later). The result is shown in the right panel of Figure 1.

```
R> mycache <- buildScoreCache(data.df = asia, data.dist = distrib,
+   max.parents = 4, dag.retained = ~ LungCancer|Smoking)
R> mp.dag <- mostProbable(score.cache = mycache)
R> fabn <- fitAbn(object = mp.dag)
```

To structurally compare two BNs, one typically uses the so called Hamming distance (De Jongh and Druzdzel 2009) which is the number of arcs that need to be changed on the reference network to exactly map to the index network. The unconstrained model has a Hamming distance of two compared to the true network, whereas for the constrained model this is unity.

The function `fitAbn()` computes the parameter estimates from a given structure, a list of distributions and a dataset. The `asia` dataset is made of Bernoulli variables and the

parameter estimates are log odds ratios. The method used is a Bayesian approach, thus the estimates are the modes of the posterior distribution.

```
R> fabn
```

The ABN model was fitted using a Bayesian approach. The estimated modes are:

```
$Asia
```

```
Asia|(Intercept)
      -4.77
```

```
$Smoking
```

```
Smoking|(Intercept)
      0.012
```

```
$Tuberculosis
```

```
Tuberculosis|(Intercept)
      -4.72
```

```
$LungCancer
```

```
LungCancer|(Intercept)      LungCancer|Smoking
      -4.28                    2.26
```

```
$Bronchitis
```

```
Bronchitis|(Intercept)      Bronchitis|Smoking
      -0.85                    1.78
```

```
$Either
```

```
Either|(Intercept) Either|Tuberculosis      Either|LungCancer
      -11.4                19.1                21.1
```

```
$XRay
```

```
XRay|(Intercept)      XRay|Either
      -3.09                8.30
```

```
$Dyspnea
```

```
Dyspnea|(Intercept) Dyspnea|Bronchitis      Dyspnea|Either
      -2.08                3.31                2.19
```

Number of nodes in the network: 8.

We interpret the output for the variable `LungCancer`. There is an arrow from node `Smoking` to node `LungCancer`. The function `fitAbn()` returns an intercept which could be interpreted as noise and an odds ratio of 2.26 on the logit scale. In exponentiating it we get the classical odds ratio (OR: 9.61) which is a measure of association between the two random variables. The OR is larger than 1 meaning that `LungCancer` and `Smoking` are positively associated. However, such OR are not biologically realistic.

In this simple example we have plenty of observations to estimate 15 parameters. In more realistic cases we have to control for over-fitting. This will be further discussed in Section 4.4. A warning for potential future users is that the R package **abn** may be computationally slower than other BN packages, which is part of the cost for the increased flexibility offered. For example, structure learning with parameter estimation in the *asia* example using the R package **abn** took approximately 2.6 seconds, in comparison using the R package **bnlearn** took less than 0.1 seconds. Thus the R package **abn** may be less suited to very large scale real world datasets.

## 1.2. Alternative R packages

Not too surprisingly, there are several R packages available for BN modeling, which often target a particular modeling or data niche. These R packages can be divided into two broad classes: those targeting parameter and structure learning and those targeting inference. Below is a selected review of the available computing libraries related to graphical modeling.

The following R packages target parameter and structure learning. The most popular R package is **bnlearn** (Scutari 2010). It implements most BD scores, in addition to information theoretic scores for continuous and discrete mixed variables. Additionally, **bnlearn** has implementations of multiple network structure learning via multiple constraint-based and score-based algorithms. This is the most versatile R package for BN modeling and should be the preferred primary software choice. When focusing on a causal framework, the R package **pcalg** is very popular (Kalisch, Mächler, Colombo, Maathuis, and Bühlmann 2012). It contains an implementation of the PC-Algorithm that outputs an essential graph. When dealing with discrete Bayesian networks only, the R package **catnet** (Balov and Salzman 2022) allows parameter and structure learning using likelihood-based criteria. The R package **deal** (Böttcher and Dethlefsen 2003) enables learning with a mixture of continuous and/or discrete variables under the conditional Gaussian distribution (a restriction that discrete nodes cannot be children of continuous nodes). Other useful R packages are actively maintained on CRAN, but none of them implement scoring procedures that deal simultaneously with multiple exponential family representatives in a Bayesian or likelihood-based framework and allow correction for grouping.

When targeting inference the most used R packages are **gRbase** (Dethlefsen and Højsgaard 2005) and **gRain** (Højsgaard 2012). These R packages do not have any structural learning algorithms, and so the BN models must be fully provided by the user, but **gRbase** and **gRain** do provide extensive prediction and inference capabilities.

The R package **abn** is the only library for BN modeling based on a fully Bayesian regression framework which can also incorporate random effects in structure and parameter learning. From a data analysis perspective this feature is very appealing as many real world datasets have correlated observations as a result of natural grouping characteristics. A unique technical feature of **abn** is that it contains an implementation of the exact search algorithm from (Koivisto and Sood 2004). The main use cases envisaged for **abn** are in relation to systems epidemiology by providing a user with outputs to aid in the understanding and interpretation of complex biological data. The Task View: Graphical models in R (Højsgaard 2021) gives a comprehensive overview of the different computing libraries available on CRAN.

Aside from R, multiple implementations of BN modeling are available in other computing environments such as **Weka** (Bouckaert 2008, which is accessible in R via the R package **RWeka**

by Hornik, Buchta, and Zeileis 2009). A popular implementation toolbox in MATLAB is the Bayes Net Toolbox (BNT, Murphy 2001). A Python implementation is the **pomegranate** library (Schreiber 2018) which could be used to perform inference in general mixture models, hidden Markov models, and Bayesian networks. A C++ implementation is the **libDAI** library (Mooij 2010) which has various (approximate) inference methods for discrete graphical models. Multiple commercial offerings of BN software are also available, such as the **Hugin** Decision Engine produced by **Hugin Expert** (<https://www.hugin.com/>, Madsen, Lang, Kjærulff, and Jensen 2003). An R package exists which interfaces to the **Hugin** Decision Engine: **RHugin**. Further examples are **BayesFusion** (<https://www.bayesfusion.com/>), **Netica** (<https://www.norsys.com/>) and **BayesiaLab** (Conrady and Jouffe 2013).

## 2. Methodological background

This section describes the theoretical foundations of ABN. First, we present the BN modeling paradigm. Then we extend this framework to describe the ABN methodology. Finally, we embed the ABN methodology into a learning scheme.

### 2.1. Bayesian network

The idea of studying observational data using a BN is quite old (Pearl 1985). Formally, a BN for a set of random variables  $\mathbf{X} = \{X_1, \dots, X_n\}$  is a directed acyclic graph (DAG) where the nodes are the random variables and the directed links are the statistical dependencies between the nodes. A directed graph  $G$  is the union of two sets: the set of *nodes* or *vertices* and the set of *arcs* or *directed edge* or *arrows*. Thus:  $G = (\mathbf{V}, \mathbf{E})$ , where  $\mathbf{V}$  is a finite set of *vertices* and  $\mathbf{E}$  is a finite set of *directed edges*. An index node  $X_j$  is said to be the parent of a node  $i$  if the set of directed edges  $\mathbf{E}$  contains a directed edge from  $j$  to  $i$ . A set of parents for a node  $j$  is denoted by  $\mathbf{Pa}_j$ . Conversely, one can easily define the set of children  $\mathbf{Ch}_j$  for a node  $j$ .

In a BN, the factorization of the joint probability distribution  $P(\mathbf{X})$  through a so-called set of local probability distributions is given by the *Markov property*, which implies that an index node  $X_j$  is dependent solely on its set of parents  $\mathbf{Pa}_j$

$$P(\mathbf{X}) = P(X_1, \dots, X_n) = \prod_{j=1}^n P(X_j | \mathbf{Pa}_j). \quad (1)$$

In Equation 1, the total number of nodes is denoted by  $n$ . A BN model,  $\mathcal{B}$ , is the union of the structure  $\mathcal{S}$  and the model parametrization  $\theta_{\mathcal{B}}$ :  $\mathcal{B} = (\mathcal{S}, \theta_{\mathcal{B}})$ . The arcs represent both *marginal* and *conditional dependencies*. Collectively, they define the structure or network which encodes the conditional independence through graphical separation. Two nodes in a BN are graphically separated or d-separated if all paths are blocked between the nodes (for a more formal explanation see Bang-Jensen and Gutin 2008; Pearl 2009). It exists operational rules to graphically detect blocked paths in a BN. Verma and Pearl (1988) show that if two nodes are d-separated by a set of nodes, then the random variables are conditionally independent given the set of variables. This theorem is the starting point of a class of BN learning algorithms called constraint-based approaches.

Another approach is based on the parametrization of the local probability distribution. In a BN, each node  $X_j$ , with parent set  $\mathbf{Pa}_j$ , is parametrized by a local probability distribution:

$P(X_j \mid \mathbf{Pa}_j)$ . The choice of the parametrization is the source of an alternative class of learning algorithms called score-based approaches. Two networks are said to be equivalent if they represent the same set of conditional independence statements. A scoring system is said to be score equivalent if it assigns the same score to equivalent networks. Score equivalence is a key notion for the score-based approach.

The Markov blanket (MB) of a node is the set of parents, children and co-parents, i.e., the parents of the specified children. An interesting property of the MB is that this is the set of nodes that fully inform an index node. For example, in the right network of Figure 1, the MB of index node `Dyspnea` consists of nodes `Bronchitis` and `Either` (e.g., `mb(fabn$abnDag$dag, "XRay")`), whereas the MB of index node `Either` consists of nodes `XRay`, `Dyspnea`, `Tuberculosis`, `LungCancer` and `Bronchitis`. A node is conditionally independent of any non-descending node in a BN given its parents.

## 2.2. Additive Bayesian network formulation

An ABN model,  $\mathcal{A}$ , is a graphical model that extends the usual generalized linear model (GLM) to multiple dependent variables through the factorization of their joint probability distribution (Lewis and Ward 2013). An  $\mathcal{A}$  model assumes that each node is a GLM where the covariates are the parents and the distribution depends on the index node. Currently, the available data distributions are `binomial(link = logit)`, `Gaussian(link = identity)`, `Poisson(link = log)` and `multinomial(link = logit)` where the last distribution is available for MLE (maximum likelihood estimation) fitting only. The default link functions are hard coded.

Given an index node  $X_j$ , a set of parents  $\mathbf{Pa}_j$ , and using the classical notation for the exponential family parametrization (Pitman 1936)

$$P(X_j \mid \mathbf{Pa}_j) = \exp(\eta(\theta_j)T(X_j, \mathbf{Pa}_j) - A(\theta_j))dH(X_j, \mathbf{Pa}_j), \quad (2)$$

where the functions  $\eta$ ,  $T$ ,  $A$ ,  $H$  may be node-dependent (the indices are omitted to simplify the notation) and where the parameters  $\theta_j$  incorporate the configuration of the parents' node. For example, in the case of binary variables, i.e.,  $X_j \in \{0, 1\}$ , Equation 2 is simplified when using the classical logit link function to

$$P(X_j = 1 \mid \mathbf{Pa}_j) = \text{logit}^{-1}(\theta_j) = \text{expit}(\theta_j),$$

resulting in the classical logistic regression model. The multinomial random variables are fitted using neural network estimator with a softmax link function.

The R package **abn** is situated in a small data analysis niche. The ABN implementation targets datasets which contain more than one type of variable out of continuous, discrete and count. The main focus is that the final model's parameters should be biologically interpretable. An  $\mathcal{A}$  is called additive in the sense that the expected change in the index node with respect to the set of parents is assumed to be additive on the exponential family link scale. Other Bayesian marginal likelihood scores, such as BDe do not assume additivity. As a direct consequence of assuming additivity, an  $\mathcal{A}$  does not model interactions in the data, which could become problematic from a biological perspective if interactions are expected. Moreover, as this technique relies on the exponential family to parametrize the model, it implies rather bold assumptions in terms of the number of parameters and thus fitting ability. Indeed, if we



simply consider a single binary variable with  $k$  parents (and no other edges) then the number of parameters in an  $\mathcal{A}$  model will scale linearly with  $k$ , while the saturated model scales as  $2^k$ , e.g. with 7 parents,  $p(X_j | \mathbf{Pa}_j)$  has 8 parameters versus 128 for the saturated model also known as conditional probability table.

There are some preliminary attempts to include statistical interactions in an ABN model as proxy for biological interactions from the data. Indeed, statistical interactions do not necessarily overlap with biological interactions. Much care should be taken when considering statistical interactions, as in a fully automated method that requires to estimate a large amount of models, adding blindly interaction terms will augment massively the computation time. Hence, the only viable solution is some post processing adjustment of the model driven by prior field-specific knowledge. In a parametric bootstrapping step, this limitation could, however, become very problematic as it could simplify the complexity of the data and discard important data specificity when the purpose was originally to prune the model in keeping all important data features.

### 2.3. Learning algorithm

Many different learning strategies have been proposed depending on the specific research question of interest and the ultimate goal of the modeling. To perform inference in BNs, one needs a probabilistic model in order to compare networks and a search algorithm to select the optimal structure. If we restrict ourselves to a purely non-dynamic network (dynamic networks happens when using either temporally dependent data or a dynamical network) and observational data, we can propose two main strategies. The first is constraint-based, where one learns the BN using statistical independence tests. The optimal network is identified using the reciprocal relationship between graphical separation and conditional independence (Spirtes 2001). The second is based on structural scoring, where each candidate network is scored and the network which has the highest score is kept. In practice, this is computationally intractable for the typical number of variables involved in a research project. The number of possible networks is massive and increases super-exponentially with the number of nodes (Robinson 1977). A practical workaround is to use a decomposable score, i.e., a score that is additive in terms of the network's nodes and depends only on the parents of the index node. This approach is very close to classical model selection in statistics (Zou and Roos 2017).

The scoring approach paradigm requires that the scores represent how well the structure fits the data (Zou and Roos 2017). Many scores have been proposed for discrete BNs in a Bayesian context (Daly, Shen, and Aitken 2011) which aim to maximize the posterior probability. Indeed, Heckerman, Geiger, and Chickering (1995) propose the so-called Bayesian Dirichlet (BD) family of scores which use a Dirichlet prior. The BD family regroups the K2, BDeu, BDs and BDla scores (Scutari 2018). For continuous BNs with multivariate Gaussian data, using an inverse Wishart prior leads to the BGe score (Consonni and La Rocca 2012). By analogy, these scores can be adapted to a mixture of variable types through using information theoretic scores. Information theoretic scores are less suitable from a theoretical perspective but more polyvalent regarding the possible data distribution. Scores within a frequentist framework have been proposed (Daly *et al.* 2011), such as Bayesian information criterion (BIC), Akaike information criterion (AIC) and minimum description length (MDL). They all have in common a goodness-of-fit component and a penalty component for model complexity. A direct and natural extension of this idea, implemented in the R package **abn**, uses the

posterior score in a Bayesian regression settings. When applied to BN scoring, it is called the marginal likelihood (MacKay 1992), as the likelihood is marginalized over the parameter space with the parameter prior acting as a penalty term.

The model learning phase involves two parts: 1. Network, skeleton or structure learning  $\mathcal{S}$ ; and, 2. Parameter learning where the model parameter is  $\theta_{\mathcal{B}}$ . Hence, in a Bayesian framework, constructing an ABN model  $\mathcal{A}$  given a set of data  $\mathcal{D}$  can be written as

$$P(\mathcal{A} | \mathcal{D}) = \underbrace{P(\theta_{\mathcal{A}}, \mathcal{S} | \mathcal{D})}_{\text{model learning}} = \underbrace{P(\theta_{\mathcal{A}} | \mathcal{S}, \mathcal{D})}_{\text{parameter learning}} \cdot \underbrace{P(\mathcal{S} | \mathcal{D})}_{\text{structure learning}} .$$

The two learning steps are inter-connected and efficient algorithms have been proposed for each. In order to learn the relationships between variables, the conditional probability distributions should be fitted. In a frequentist setting and for Gaussian, Bernoulli and Poisson distributed variables, this can be done using the classical iterative re-weighted least squares (IRLS) algorithm (Faraway 2016). The multinomial random variables are fitted through a softmax function with the maximization of the conditional likelihood. The structure selection step can be done using a heuristic or exact approach.

A useful practical feature of the ABN methodology is the ability to impose external expert knowledge. Indeed, in most applied data analyses, some part of the network is likely known. For example, if two random variables are temporally related, the direction of the possible arrow is known. Or based on existing literature a possible connection is known to be expected. The R package **abn** allows such external causal input through using two matrices, one in which arcs that are not allowed (banned) are defined, and a second matrix which defines those arcs which must be present (retained) in all networks considered during the structural learning process. These matrices are used to compute the list of valid parent combinations. A theoretically more rigorous approach, as suggested by Heckerman *et al.* (1995), is to augment the observed data with synthetic data that represents one's causal beliefs. The practical feasibility of this in an ABN analysis remains an open question (McCormick, Sanchez-Vazquez, and Lewis 2013).

### 3. The R package **abn**

We now describe the functionality of the R package **abn** and provide insights into how some of the key functions are implemented. We also briefly present simulation results comparing parameter estimates using Bayesian and MLE approaches, and the performance of different scores when attempting to recover the true network.

The R package **abn** has three levels of functionality, each with a different purpose:

- *Core functions*: Aimed at performing an ABN analysis (scoring and learning)
- *Ancillary functions for analysis*: Aimed at supporting analyses through enhanced plotting abilities, accounting for uncertainty in structure through link strength estimation, and comparing structures or retrieving structural metrics
- *Ancillary functions for simulation*: Aimed at helping to simulate ABN models by simulating DAGs and generating data.

A classical ABN analysis is the sequential application of three R functions, see Section 1.1 (typically: `buildScoreCache()`, `fitAbn()` and `mostProbable()`). The analysis is divided into three functions in order to give more freedom to the end user, through for example, the ability to tune parameters or give more refined control over possible learning or fitting issues. All R function names in **abn** are unified and are using exclusively CamelStyle. Older names are deprecated.

### 3.1. Set of core functions

The two main core functions for scoring ABN models are `buildScoreCache()` and `fitAbn()`. The former is essentially a loop over an minimalistic version of the latter to score all admissible atomic networks. The minimalistic version of `fitAbn()` (inside `buildScoreCache()`) is fast and return only a network score. The enhanced version of the scoring function, i.e., `fitAbn()` is used to score a DAG and to return parameter estimates. These functions have a Bayesian and an MLE implementation which are not equivalent in their output (there is an argument `method` that can be "bayes" or "mle" to choose the implementation). They require minimally a named dataset, a named list of the nodes' distributions and an upper limit for network complexity. `buildScoreCache()` first computes an empty list of all valid parent combinations using banning and retaining input matrices (which are assumed to be empty by default). Then it iterates through the cache to score each candidate piece of the network. In the Bayesian implementation, `buildScoreCache()` and `fitAbn()` estimate a Bayesian regression using the following parameter priors: weakly informative Gaussian priors with mean 0 and variance 1000 for each of the regression parameters of the model (both binomial and Gaussian), as well as diffuse Gamma priors (with shape and scale equal to 0.001) for the precision parameters in Gaussian nodes in the model using internal INLA code.

`buildScoreCache()` within an MLE setting uses an IRLS algorithm depending on the given list of distributions at each step of the scoring process. For the special case of binomial nodes, the usual logistic regression is tried. If this fails to find an estimate a bias-reduced tailored algorithm is used (Firth's correction), which is known to improve the accuracy of regression coefficients in the presence of separation (Van Smeden *et al.* 2016). If the algorithm still fails to find a finite estimate then some predictors are sequentially removed until the design matrix becomes fully ranked. These three steps ensure the R package **abn** is able to score a dataset even if there is (quasi) data separation. Multinomial random variables can also be estimated through an IRLS-like algorithm. A well known weakness of this approach is the very high per-iteration cost due to sparsity of the intermediate matrices. A robust, easy to implement and fast solution to estimate unregularized multinomial regression models is to use neural networks with no hidden layers, no bias nodes and a softmax output layer. The optimization is done through maximum conditional likelihood. This procedure is implemented in R by the **nnet** package (Venables and Ripley 2002) and is the chosen solution for multinomial estimation in **abn**.

The `fitAbn()` function scores a given network. It requires a valid DAG, a named dataset and a named list of distributions. It returns the list of scores for each node, the parameter estimates, the standard deviation and the  $p$  values. Special care should be taken when interpreting and displaying the  $p$  values. Indeed, the DAG has been selected using the goodness of fit metric, so at least some form of adjustment for multiplicity should be used.

A unique feature of `fitAbn()` and `buildScoreCache()` is the possibility to take one-layer clustering into account. In some situations data collection has a clear grouping structure

which raises the potential issue of correlated data points. This can lead to over-dispersion and analyses which are over-optimistic, as the true level of variation in the data is underestimated. This implies introducing an additional correlation structure via random effects, i.e., adjusting for correlated residuals or non-constant variance. Thus, each node becomes a Bayesian generalized linear mixed model (GLMM, [Breslow and Clayton 1993](#); [Faraway 2016](#)) rather than a Bayesian GLM (see [Section 4.6](#) for more details). The rationale for using Bayesian GLMMs instead of Bayesian GLM is that standard errors of the coefficients are underestimated during the structural search and so the DAGs produced without random effects are anti-conservative – have more rather than less structure. This possible excess of structure could be trimmed during the (later) MCMC part of the analysis if it is not well supported by the data.

We compute the posterior distribution of the ABN model computed above and check if they widen, i.e., widening between ABN models with and without clustering. If this is the case then we must take clustering into account in the scoring scheme. From an applied perspective, the major limitation of including random effects is the additional computational burden of this approach. Indeed, if adjustment for clustering is unlikely to impact the estimates, the additional model flexibility needs to be weighted with the additional computational time. Without adjustment for clustering, the model is then much simpler and parsimonious. The clustering adjustment could eventually be performed on a subset of the nodes.

For learning a BN two types of algorithms are implemented: exact and heuristic searches. The exact procedure, `mostProbable()`, runs the exact order-based structure discovery approach of [Koivisto and Sood \(2004\)](#) to find the most probable network. Its input is a cache of pre-computed scores from `buildScoreCache()`, the desired score and structural prior to use. As described in [Koivisto and Sood \(2004\)](#), `mostProbable()` uses dynamic programming to marginalize orders analytically. It cannot handle large numbers of nodes, e.g. more than 25 or so nodes, but on similar sized problems it outperforms MCMC and probabilistic approaches. The exact search is implemented with two structural priors: the Koivisto prior which states that different cardinalities of parents are considered to be equally likely a priori; and a structurally uninformative prior where parent combinations of all cardinalities are equally likely. The heuristic procedure, `searchHeuristic()` are also based on hill-climber, Tabu and simulated annealing ([Scutari 2010](#)). This function requires a cache of pre-computed scores, the desired score and some method-dependent arguments.

### 3.2. Defining a DAG

To specify a DAG, the R package **abn** uses an adjacency matrix, i.e., a named square matrix with an entry in the  $ij$ th cell if there is a directed edge from  $j$  (parent) to  $i$  (child).

The R package **abn** also recognizes a formula-like syntax, similar to the classical R functions `lm()`, `glm()`, etc. A typical formula is `~ node1|parent1:parent2 + node2:node3|parent3`. The formula statement has to start with `~`. In this example, `node1` has two parents (`parent1` and `parent2`). `node2` and `node3` have the same `parent3`. The parent's names have to exactly match those given by the argument `data.df`. The symbol `:` is the separator between either children or parents, the symbol `|` separates children (left side) and parents (right side), the symbol `+` separates terms and `.` replaces all the variables in `data.df`. Due to this feature, those symbols cannot be used in the names of the random variables. The terms in the formula are additive in the sense that the nodes could appear multiple time on the left side. The banned matrix could be produced by the following statement `~ node1|. ,` which would

prevent the variable `node1` from having any parents. Currently it is not possible to incorporate interaction terms in this formula statement.

### 3.3. Ancillary functions for analysis

The functions `plotAbn()` and `toGraphviz()` are useful for plotting DAGs. `plotAbn()` can display a DAG with fitted values and arrows with thickness proportional to arc strength.

The concept of link strength for discrete BNs was introduced in Boerlage (1992) and is a useful concept to measure the level of connection of any pair of nodes in a discrete BNs. A good unpublished overview is given by Ebert-Uphoff (2009). Link strength is helpful when interpreting a discrete BN model as it helps to account for uncertainty given that in a DAG an arc is either present or absent. This strong dichotomization of structural relationships in BNs is often hard to interpret. Link strength tends to give a proxy for arc support by the data and from an applied perspective can be interpreted in a broadly similar manner to regression coefficients. In practice, the function `linkStrength()` discretizes the dataset using a large choice of histogram rules to compute multiple link strength metrics. Then, the estimates of the empirical distribution are plugged into the definition of the entropy to return the so-called empirical entropy. A well-known problem of empirical entropy is that the estimations are biased due to sampling noise. It is also known that the bias decreases as the sample size increases. The mutual information which is a measure of the mutual dependence between two nodes is computed from the observed frequencies through a plug-in estimator based on the entropy. The link strength in `abn` could be computed based on the mutual information, the correlation or the statistical distance.

Two functions are useful for comparing DAGs. The `compareDag()` function returns several graph metrics for comparing two DAGs: the confusion matrix, or also called error matrix in the machine learning literature (Stehman 1997). `compareDag()` provides a list with the true positive rate, the false positive rate, the accuracy, the G-measure, the F1-score, the positive predictive value, the false omission rate and the Hamming-Distance. The `infoDag()` returns a list of standard metrics for describing a DAG that contains the number of nodes, the number of arcs, the average Markov blanket size, the neighborhood average set size, the parent average set size and children average set size.

The R function `scoreContribution()` computes the score contribution from each individual observation to the total network score and additionally returns the diagonal entries of the hat matrix. This function attempts to produce influential measures adapted to ABN models. In a regression context it is possible for a single or small group of observations to exert substantial influence on the results. It is therefore important to identify influential observations and take them into consideration when interpreting results (Belsley, Kuh, and Welsch 2005).

### 3.4. Ancillary functions for simulation

To simulate DAGs and ABN data, the functions `simulateDag()` and `simulateAbn()` are provided. The function `simulateDag()` generates DAGs with an arbitrary arc density. To ensure acyclicity, it samples a triangular adjacency matrix. The arc density is tuned with a binomial sampling probability. The `simulateAbn()` generates ABN data using the R package `rjags` (Plummer 2022). Simulating observations from a given structure is done with a random number generator, respecting the node ordering using `JAGS` (Plummer 2003). It first creates, in the index repository, a text file using the `openBUGS` or `WinBUGS` syntax called BUGS file.

Then it uses this file to simulate the data. Alternatively, a BUGS file could be provided by the user. This function produces a data frame. The primary purpose of these two functions is to assess the effectiveness of other functions in the R package **abn**, but they could also be used to plan and conceive systems epidemiology studies in assessing the necessary number of observations needed as a function of expected effect sizes.

## 4. Case study

The purpose of this case study is to perform a fully reproducible and transparent analysis of an open access observational dataset. The main objective is to produce a reliable network together with the necessary information to allow applied researchers to interpret and report it. The final model is controlled for over-fitting, and some strategies for controlling for clustering are presented. The code for the analysis is provided to help disseminate the ABN approach in the systems epidemiology community by addressing every step from an applied and interpretative perspective. The proposed procedure for performing an ABN analysis could be quite complex for a new user of **abn**. The left panel of Figure 2 shows the workflow diagram of a typical ABN analysis. It is the sequential application of three functions: `buildScoreCache()` for computing a cache of pre-computed scores; a search algorithm such as `mostProbable()` or `searchHeuristic()`; and `fitAbn()` for fitting the final model to the data. In the right panel, the bootstrapping workflow is presented. It is very similar to a typical ABN analysis, except that it is based on simulated datasets to control for possible over-fitting.

### 4.1. Data description and library loading

The data to be analyzed were collected in March 1987 and comprise of growth performance and abattoir findings from the commercial production of pigs across 15 Canadian farms (Dohoo, Martin, and Stryhn 2003). The research question is whether inter-relationships exist among two respiratory diseases (atrophic rhinitis and enzootic pneumonia), ascarid level and daily weight gain. This case study is well suited for demonstrating the unique ability of BN modeling to disentangle complex relationships from observational data. The data used here is a simplified version of the original dataset (some variables have been discarded). It consists of 341 observations of the 9 variables described in Table 2. A nice feature of this dataset is that it is composed of continuous, discrete and count distributed variables. Furthermore, these data have a natural grouping structure as animals come from different farms. This allows us to showcase the capabilities of the R package **abn** in controlling for a grouping effect.

Once the R package is installed and loaded, the dataset `adg` is available (via lazy data mechanism).

```
R> library("abn")
R> dim(adg)

[1] 341  9

R> head(adg, 2)

  AR pneumS female livdam eggs wormCount age adg farm
1  1      0      0      1      0          0 196 453   4
2  1      0      0      1      0          0 175 501   9
```

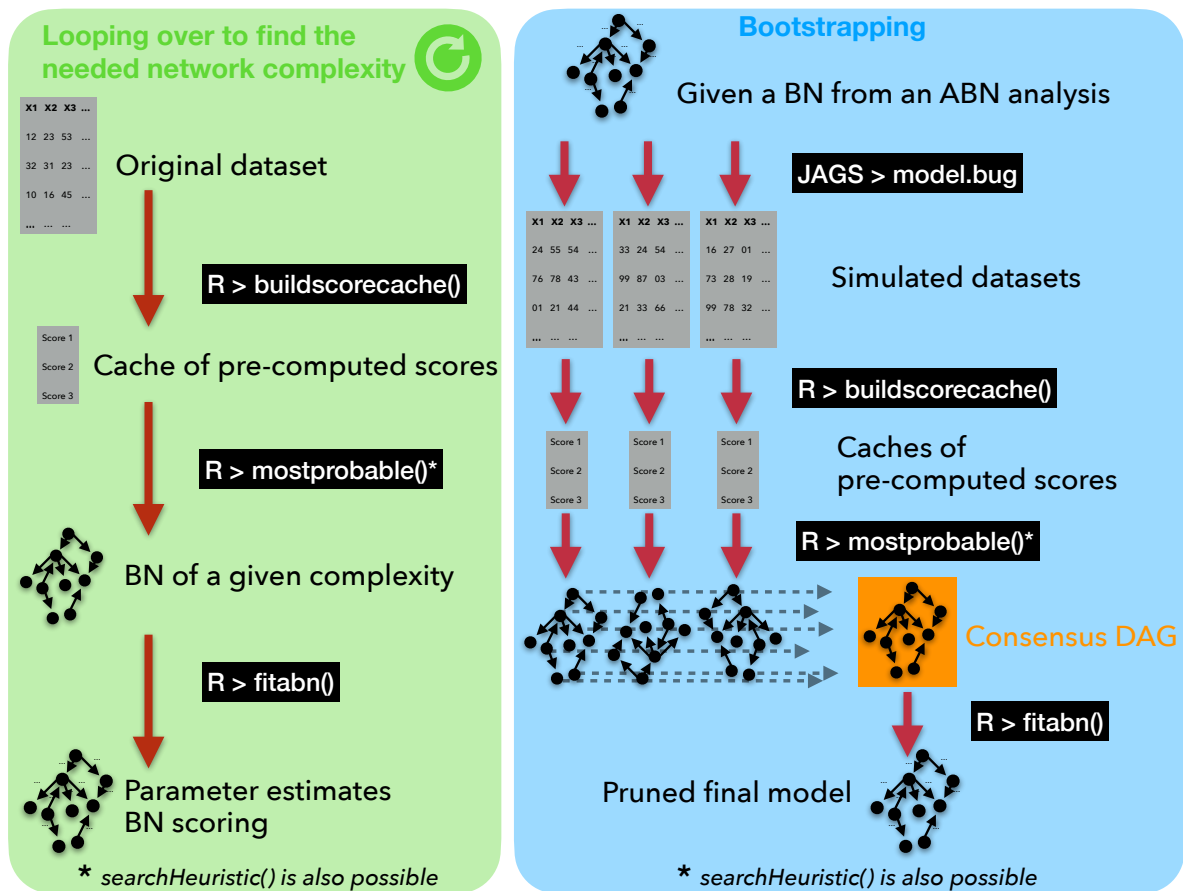


Figure 2: Schematic workflow diagram for a typical ABN analysis. The left panel is the typical workflow for an ABN analysis. The right panel describes the bootstrapping procedure.

Variable	Meaning	Distribution
AR	Presence of atrophic rhinitis	Binomial
pneumS	Presence of moderate to severe pneumonia	Binomial
female	Sex of the pig (1 = female, 0 = castrated)	Binomial
livdam	Presence of liver damage (parasite-induced white spots)	Binomial
eggs	Presence of fecal/gastrointestinal nematode eggs at time of slaughter	Binomial
wormCount	Count of nematodes in small intestine at time of slaughter	Poisson
age	Days elapsed from birth to slaughter (days)	Continuous
adg	Average daily weight gain (grams)	Continuous
farm	Farm ID	Discrete

Table 2: Description of the variables provided by the dataset `adg`.

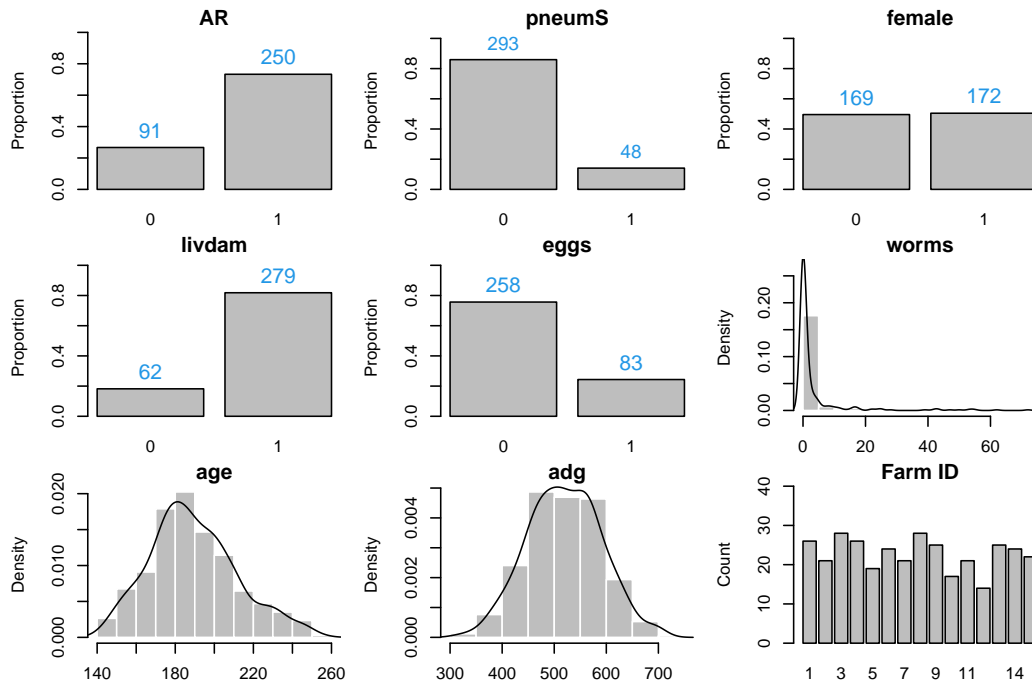


Figure 3: Descriptive distributions of the variables of the dataset `adg`.

```
R> (n.obs <- nrow(adg))
```

```
[1] 341
```

## 4.2. Data preparation

Each variable in the model needs to be associated with a distribution. Thus, one needs to create a named list that contains all the variables and the corresponding distributions. As a reminder, the available data distributions are binomial, Gaussian, Poisson and multinomial, where the last distribution is available in the R package **abn** for MLE fitting only. For the first analysis we do not include the grouping variable `farm` and we coerce binary variables to factors.

```
R> dists <- list(AR = "binomial", pneumS = "binomial", female = "binomial",
+   livdam = "binomial", eggs = "binomial", wormCount = "poisson",
+   age = "gaussian", adg = "gaussian")
R> df <- adg[, which("farm" != names(adg))]
R> df[, 1:5] <- lapply(df[, 1:5], function(x) factor(x))
```

A nice feature of the R package **abn** is the ability to input prior information about structural beliefs to guide the structural search. Indeed, prior expert knowledge is common in systems epidemiology. In this case study it is reasonable to assume that none of the variables in the model are going to affect the sex of the animal, an inborn trait. To encode this information, we ban all the arcs going to `female` by setting their value to 1 (banned), opposite to the



default value 0 (not banned) in an adjacency matrix-like formulation. The rows are children and the columns are banned parents of the index nodes.

```
R> banned <- matrix(0, ncol(df), ncol(df))
R> colnames(banned) <- rownames(banned) <- names(df)
R> banned["female", which("female" != names(df))] <- 1
R> banned
```

	AR	pneumS	female	livdam	eggs	wormCount	age	adg
AR	0	0	0	0	0	0	0	0
pneumS	0	0	0	0	0	0	0	0
female	1	1	0	1	1	1	1	1
livdam	0	0	0	0	0	0	0	0
eggs	0	0	0	0	0	0	0	0
wormCount	0	0	0	0	0	0	0	0
age	0	0	0	0	0	0	0	0
adg	0	0	0	0	0	0	0	0

An equivalent formula statement is `~female|.`, because all the variables of the data frame are used in the model.

By default, the R package **abn** assumes no banned or retained arcs. See Section 3.2 on how to specify banned or retained arcs by using a formula-like syntax.

### 4.3. Structure search

For computational reasons it is advised that the maximum number of parents allowed for each node be constrained. We start by computing a cache of pre-computed scores with a single parent per node and subsequently increase the number of parents until the network score of the optimal structure does not get any larger, even when more parents are allowed. Based on the cache of pre-computed scores, an exact search using the function `mostProbable()` is performed and the network score is computed. In R this is done using a simple `for` loop with the functions `buildScoreCache()`, `mostProbable()` and `fitAbn()`.

```
R> result <- list()
R> for (i in 1:7) {
+   mycache <- buildScoreCache(data.df = df, data.dists = dists,
+     dag.banned = banned, max.parents = i, method = "bayes")
+   mydag <- mostProbable(score.cache = mycache)
+   result[[i]] <- fitAbn(object = mydag)
+ }
R> result.mlik <- sapply(result, function(x) x$mlik)
```

Figure 4 displays the network score as a function of the number of parents. The maximum log marginal likelihood ( $-2709.25$ ) is achieved with 4 parents. As one can see, the network score increases quickly at the beginning but then plateaus. A strong assumption is to say that if increasing the network complexity by one does not change the network score, then the maximum was achieved. Indeed, nothing prevents the maximum network score being

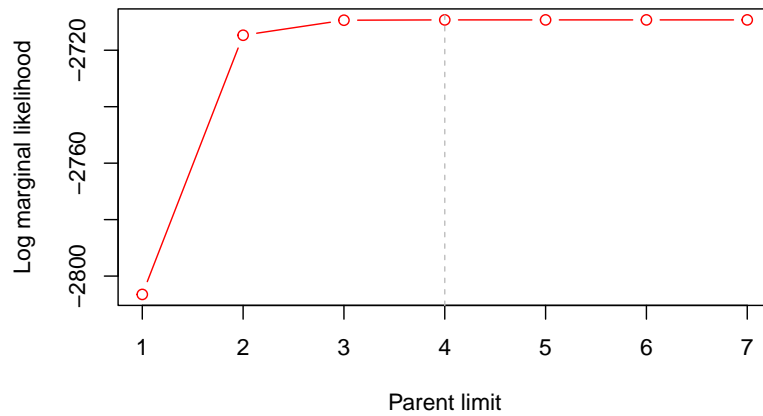


Figure 4: Total Bayesian network log marginal likelihood as a function of the number of parents.

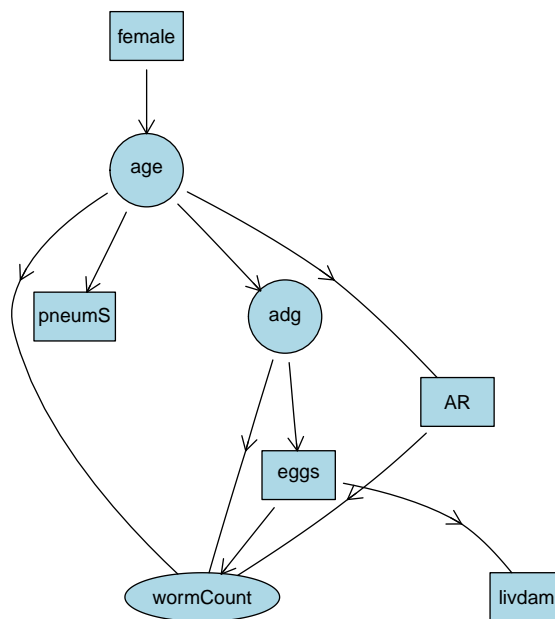


Figure 5: DAG selected using an exact search with a model complexity of four parents.

constant when increasing the number of parents but still changing further when increasing again.

In the previous code chunk, instead of the `mostProbable()` function call, a heuristic search provided by `searchHeuristic()` is possible. By nature, the approach does not guarantee that the network identified has an optimal network score conditional on network complexity nor that it is actually a (local) optimum. Of course it is also possible to run an analysis with a pre-defined network complexity.

Figure 5 shows the DAG selected using `mostProbable()` with a model complexity of 4 parents.

This DAG has 10 arcs for 8 variables, so this is a relatively sparse model (the average number of parents per node is 1.25, the average size of the Markov blanket is 3.75 and each node has on average 2.5 neighbors; as returned by function `infoDag()` or `summary()`). The square nodes are Bernoulli distributed, the oval nodes are normally distributed and the diamond node is Poisson distributed.

```
R> max.par <- which(result.mlik == max(result.mlik))[1]
R> fabn <- result[[max.par]]
R> unlist(infoDag(fabn))
```

n.nodes	n.arcs	mb.average	nh.average
8.00	10.00	3.75	2.50
parent.average	children.average		
1.25	1.25		

#### 4.4. Control for over-fitting

One major concern in BN modeling is the tendency for over-fitting. Indeed, the number of observations is generally very limited in comparison to the number of parameters in the model. The number of possible models, i.e., DAGs, increases super-exponentially with the number of random variables (Robinson 1973). The number of DAGs possible with 25 nodes is about a googol ( $10^{100}$ ).

More conceptually, the question remains of how to present the results of a BN modeling analysis. Multiple strategies have been proposed to create a summary network. One could perform multiple heuristic searches. One common and very simple approach is to produce a single robust BN model of the data mimicking 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. The function `searchHillclimber()` performs such an analysis. Rather than using the majority consensus network as the most appropriate model for data, an alternative approach would be to choose the single best model found during a large number of heuristic searches using `searchHeuristic()`, or if possible an exact search using `mostProbable()`.

Ideally, enough structural searches should be run so as to provide reasonable coverage of all the key features of the model landscape. The best model chosen should then be adjusted for possible over-fitting. Like the majority consensus network, which effectively averages over many different competing models and therefore should generally select only robust structural features, a single best DAG strategy should be controlled for over-fitting. Indeed, choosing the DAG from a single model search is far more likely to contain some spurious features, especially when dealing with small datasets of around several hundred observations. It is extremely likely to over-fit, as one can easily demonstrate using simulated data.

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 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 chosen even during an exhaustive search. Indeed, it may comprise contradictory features, a

typical risk when averaging across different models. In addition, a majority consensus network need also not to 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 when 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, element-wise matrix multiplication of the highest-scoring DAG and the majority consensus DAG gives a new DAG which only contains the structural features in both models. An alternative approach for tackling this problem, would be to use Bayesian model averaging. Such structural Markov chain Monte Carlo (MCMC) will be discussed in Section 6.

Friedman, Goldszmidt, and Wyner (1999) presented a general approach for using parametric and non-parametric bootstrapping to select BN models/DAG structures. They showed that a non-parametric approach seems to converge less rapidly in terms of number of samples but requires fewer assumptions as it relies on data and not on model. The non-parametric bootstrapping approach is relatively easy to implement and efficient in extracting robust features from data. The parametric approach can be implemented by using readily available MCMC sampling software such as **JAGS** or **WinBUGS**. The basic idea is to take the structure having the best network score and use these samplers to generate bootstrap datasets from this model. That is, independent realizations from the model which can be used to generate datasets of the same size as the observed data. Then the model search is repeated treating the bootstrap data as the observed data. Generating many bootstrap datasets and conducting searches on each allows us to estimate the percentage support for each arc in the highest-scoring model. In other words, we find out how many of the arcs in the highest-scoring model can be recovered from a dataset of the same size as that which was actually observed. Obviously, the more data are observed, the more statistical power and recoverable structural features can be learned. Arcs with a lower level of support, e.g.,  $< 50\%$ , can be pruned from the best fitting model, assuming that these are potentially a result of over-fitting. The resulting model – possibly with arcs pruned from the original model – is the chosen model for the data. The 50% threshold is arbitrary and can be adapted according to the expected arc density or model complexity.

An obvious limitation of the parametric bootstrapping method is the fact that the adjustment for over-fitting transfers the model assumptions and model limitations of the original method to the simulated data. When using a non-parametric approach the original data are used and the features not captured by the model are still present. Other options for trimming or pruning arcs exist, but without obviously superior alternatives this approach appears reasonable and a strategy commonly used in analyses of phylogenetic trees.

While parametric bootstrapping as a general technique is well established and conceptually elegant, it may in practice not be computationally feasible. Even when taking the least demanding approach of conducting only one heuristic search per bootstrap dataset, the number of datasets/searches required to get robust support values for each arc in the best fitting model may be large and well beyond what is reasonably possible even using high performance computing hardware.

#### 4.5. Parametric bootstrapping

The chosen dataset allows us to do fast parametric bootstrapping. Given a BN model – a

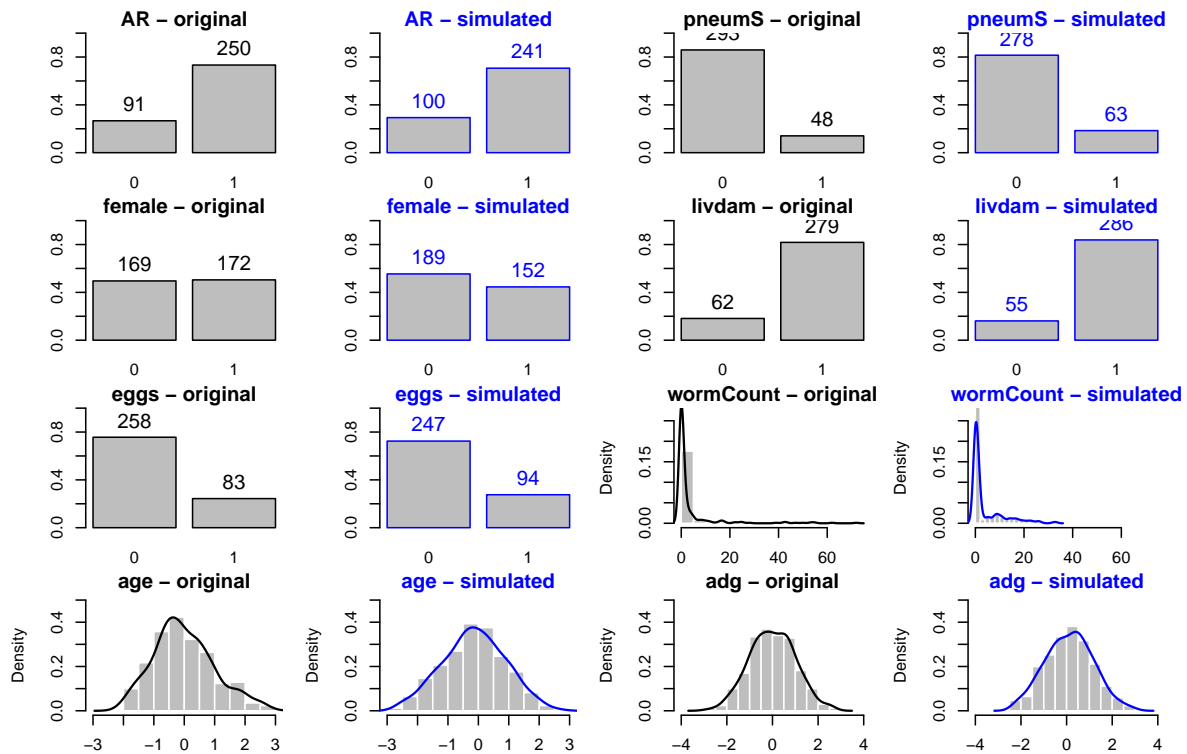


Figure 6: Example of simulated data (in blue) versus original data (in black).

DAG structure plus parameter priors – the first step is to estimate the posterior parameters. The second step is to implement the DAG structure together with the posterior parameters in a BUGS file. To make the implementation as general as possible, we present an approach based on `dcat()` from **JAGS** that discretizes each posterior density across a fine grid.

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 in `fitAbn()` using a bit of trial and error. It is crucially important to give a sufficiently wide range so that all of the upper and lower tails of the distribution are included, e.g., the range should encompass where a density plot first drops to approximately zero at each tail.

Figure 6 compares the marginal distributions of the simulated data (in blue) with the original data (in black). By default, `fitAbn()` centers and standardizes continuous data. As one can see, the simulated data look fairly good except for the count data (e.g., `wormCount`), for which the long right tail is under-represented in the simulated data.

The code displayed in Appendix C performs 2500 bootstrapping iterations from a ABN model defined in a BUGS file called `model18vPois.bug` using the R package `rjags`. The ABN model is the one produced in the last paragraph. The fitted data, i.e., the modes, are the ones generated via `fitAbn()` with `compute.fixed = TRUE`. A thinning of 20 is used to reduce auto-correlation in the simulated samples. The global approach is depicted with the pseudo-code given in Algorithm 1. In the following, the list of DAGs selected from the simulated datasets is called the bootstrapped samples. In order to simulate the variables of the dataset, we need to provide a model for each of them using the aforementioned parameters estimates.

---

**Algorithm 1:** Pseudo-code for bootstrapping ABN models.

---

**Results:** A list of DAGs and bootstrapped dataset

**Input:** A fitted ABN model, an original dataset, model complexity limit, a number of desired bootstrap runs, a BUGS file model

**Output:** List of matrices and dataframe

**for**  $1:n$  **do**

    Read a BUGS file and run **JAGS** over it

    Extract simulated samples

    Format data

    Compute a cache of pre-computed scores with `buildScoreCache()`

    Run an exact (`mostProbable()`) or a heuristic search (`searchHeuristic()`)

    (optional) Fit an ABN model to the selected BN with `fitAbn()`

**end**

---

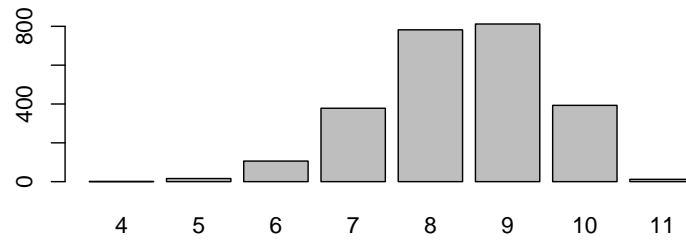


Figure 7: Histogram of the distribution of the number of arcs in the bootstrapped searches.

For instance, the binomial node `AR` in our DAG has one incoming arc coming from the node `age`. In a regression setting this would be translated into:

$$\text{logit}(\text{AR} = 1 \mid \text{age}) = \alpha + \beta \cdot \text{age} + \varepsilon,$$

where  $\alpha$  is the intercept,  $\beta$  is the regression coefficient for variable `age`, and  $\varepsilon$  is the error term. Then, the values of  $\alpha$  and  $\beta$  are sampled using the function `dcat()` within **JAGS** from the original but discretized distribution of parameters generated by `fitAbn()`.

Figure 7 displays the distribution of the number of arcs from the selected BNs from the simulated datasets, i.e., the bootstrapped samples. The network selected from the original dataset had 10 arcs. This is an indication of over-fitting from the original model.

In order to produce the final pruned DAG (named `trim.dag` subsequently), we measure the prevalence of each arc in the generated structures and retain only arcs present in at least 50% of the bootstrapped samples.

The matrix displayed below shows the percentage of arcs retrieved within the bootstrap samples. As one can see, some arcs are clearly supported or excluded, but some are very close to the 50% threshold. It is not surprising given the typical number of observations in an epidemiological study (a few hundred). An alternative approach would be to consider an undirected graph that includes all arcs supported regardless of direction in more than 50% of the bootstrapped samples. Such an approach relies on the idea that the ABN methodology

is acausal and the existence of an arc is more important than its direction. This decision is likely to be problem-specific.

```
R> percdag <- Reduce("+", dags) / length(dags) * 100
R> trim.dag <- (percdag > 50) * 1
R> round(percdag, digits = 0)
```

	AR	pneumS	female	livdam	eggs	wormCount	age	adg
AR	0	0	0	0	0	0	76	5
pneumS	1	0	1	0	0	0	41	8
female	0	0	0	0	0	0	0	0
livdam	0	0	0	0	53	2	0	0
eggs	0	0	0	24	0	0	8	73
wormCount	70	1	0	0	100	0	100	60
age	18	17	55	0	4	0	0	26
adg	1	2	10	0	6	0	74	0

#### 4.6. Control for clustering

In the `adg` dataset, the observations were collected from different farms (random variable encoded as `farm`). As such, a workaround is needed to introduce an additional correlation structure via random effects to avoid overly optimistic parameter estimates. Three strategies for clustering adjustment are possible in an **abn** analysis. They are ranked in terms of computational complexity:

- *Adjustment at the regression phase:* Given the structure selected without adjustment (eventually controlled for over-fitting), it is possible using the function `fitAbn()` to adjust the regression coefficients for clustering.
- *Adjustment at the bootstrapping phase:* Given the structure selected without adjustment, one can alter the code aiming at controlling for over-fitting in adding a random effect into the model.
- *Adjustment at the learning phase:* Learning the structure with cluster adjusted scores with the function `buildScoreCache()`.

##### *Adjustment at the regression phase*

The function `fitAbn()` has the ability to control internally for clustering using the arguments `group.var` and `cor.var`. One can choose which variable should be adjusted. The grouping variable has 15 levels. In the function `fitAbn()`, we apply the grouping adjustment to all random variables.

The following code will produce adjusted regression coefficients. We have to include the grouping variable in the `data.df` argument.

```
R> df.gr <- adg
R> df.gr[, c(1:5, 9)] <- lapply(df.gr[, c(1:5, 9)], function(x) factor(x))
```

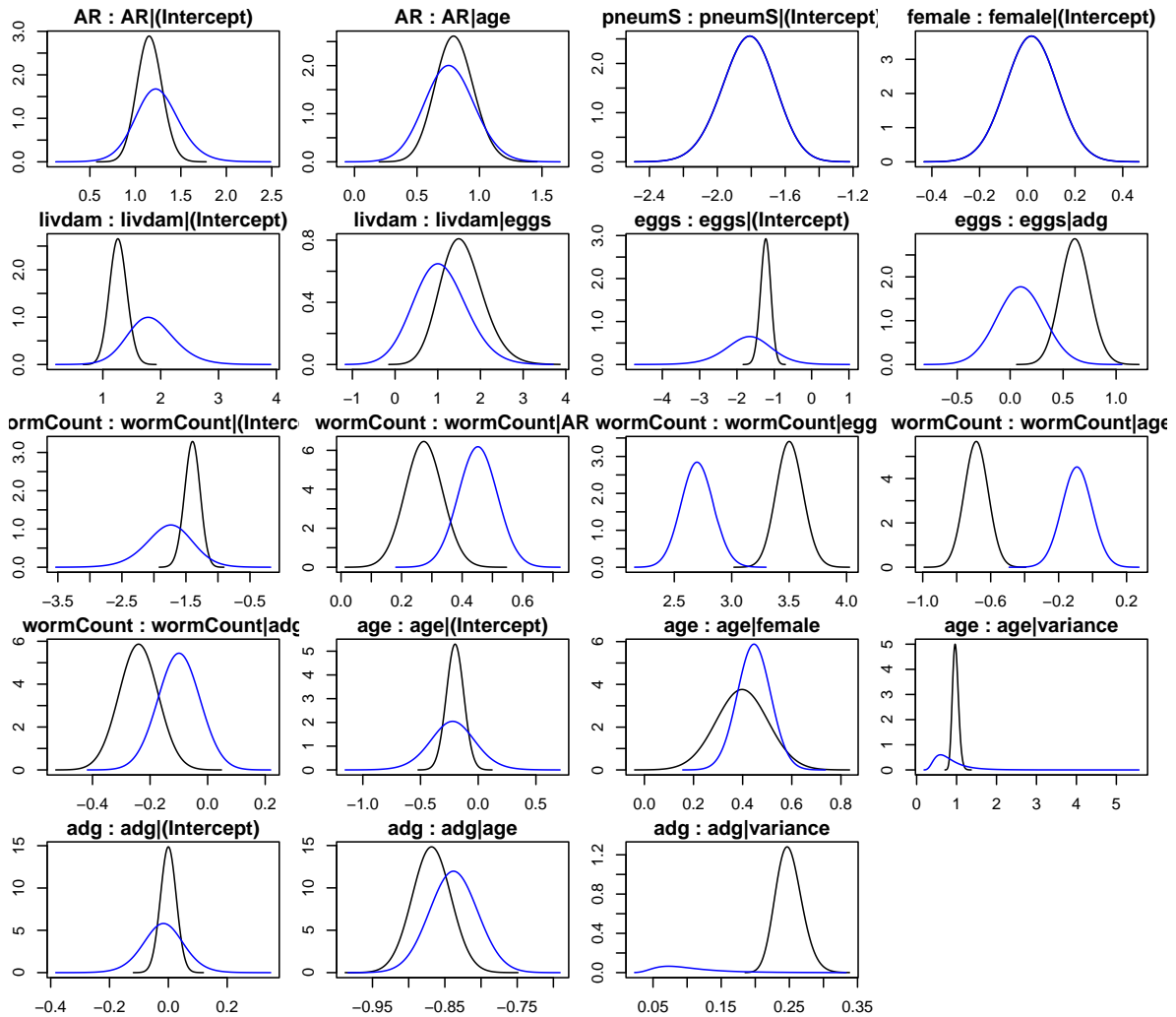


Figure 8: Marginal posterior densities of model parameter without correction for clustering in black and marginal posterior densities corrected with random effect in blue.

```
R> marg.f.grouped <- fitAbn(dag = trim.dag, data.df = df.gr,
+   data.dists = dists, cor.vars = c("AR", "livdam", "eggs", "wormCount",
+   "age", "adg"), group.var = "farm", compute.fixed = TRUE)
```

On a personal computer, computing the adjusted model takes several minutes to compute, whereas the unadjusted model ran in less than a second.

Figure 8 shows the marginal posterior densities of the model parameter corrected for clustering using random effect (in blue) and without correction (in black). The adjustment has been applied to every node except the node female, as one can see in Figure 8. Indeed, after having done the adjustment for all variables, we noticed that the random effect has been poorly estimated for the node female and thus we decided to rerun the analysis without adjustment on the variable female and including all other variables with adjustment. As one can see in Figure 8, the correction for clustering has an impact on the posterior distributions. The node `wormCount` has significantly different posterior estimates.



*Adjustment at the bootstrapping phase*

In the BUGS file used to perform bootstrapping (in the previous paragraph the BUGS file is named `model8vPois.bug`), one can add a random effect for each level of the clustering variable. The random effects are typically chosen as normally distributed with mean zero and precision parameter determined with a diffuse gamma prior. In practice one modifies the user provided BUGS file with a for loop over the number of levels of the clustering variable ( $M = 15$ ; the number of farms) for each variable that should be adjusted. A prior for the precision parameter should also be provided.

The following lines of code are an example of possible change in order to introduce random effects for the variable `adg` in the BUGS file. This code should be added for each variables that need to be adjusted for clustering.

```
$> for(j in 1:M){
$   rv.adg[j] ~ dnorm(0.0, prec.rv.adg);
$ }
$> prec.rv.adg ~ dgamma(1, 5E-05);
```

The previous line defines the prior distribution of the precision.

This approach aims at verifying which of the selected arcs are robust enough to pass the bootstrapping phase when correction is applied to account for the additional variance produced by the clustering. This approach is not implemented for the current example.

*Adjustment at the learning phase*

The adjustment could be done at the learning phase when pre-computing the network scores allowing for extra variability at the lowest possible level and represents the computationally most demanding approach. This is done using the `group.var` and `cor.var` arguments in `buildScoreCache()`. In the following code the same structural constraints have been used and the number of possible parents is limited to four.

```
R> mycache.l <- buildScoreCache(data.df = df.gr, data.dists = dists,
+   dag.banned = banned, max.parents = 4, group.var = "farm",
+   cor.vars = c("AR", "livdam", "eggs", "wormCount", "age", "adg"))
```

The cache is used in one of the search functions (without any argument modifications) and the subsequent fit by `fitAbn()` inherits the same specification as `buildScoreCache()`.

Note that fitting such a model is often quite tricky because numerical instabilities and default parameters of the `control` arguments may not be satisfactory. However, there is no better recipe than to set `verbose = TRUE` with trial-and-error of individual control arguments. For our example, it is not straightforward as well. The instability is mainly due to the strong dependency between `eggs` and zeros of `wormCounts` and the many levels of the variable `farm`.

**4.7. Accounting for uncertainty in BN models**

Calculating the so-called link strength is useful for both visualization and approximate inference, and it can be seen as a proxy for arc uncertainty. The strength of the arcs is a complementary metric to regression coefficients. We use a link strength metric called the true

	2.5 Quantile	Median	97.5 Quantile	Interpretation	PLS
AR age	1.44	2.14	3.20	Odds ratio	0.09
livdam eggs	0.88	2.83	10.52	Odds ratio	0.04
eggs adg	0.71	1.11	1.72	Odds ratio	0.13
wormCount AR	1.38	1.57	1.79	Rate ratio	0.28
wormCount eggs	11.31	14.91	19.73	Rate ratio	0.41
wormCount age	0.77	0.91	1.08	Rate ratio	0.43
wormCount adg	0.78	0.90	1.04	Rate ratio	0.38
age female	0.31	0.44	0.58	Correlation	0.02
adg age	-0.90	-0.84	-0.77	Correlation	0.39

Table 3: Summary of the marginal posterior distributions of the parameters.

average link strength percentage or percentage link strength (PLS), which expresses by how many percentage points the uncertainty in variable  $Y$  is reduced by knowing the state of its parent  $X$  if the states of all other parent variables are known (averaged over the parent states using their actual joint probability). The actual definition for an arc going from node  $X$  to node  $Y$  is:

$$\text{PLS}(X \rightarrow Y) = \frac{H(Y | Z) - H(Y | X, Z)}{H(Y | Z)},$$

where the remaining set of parents of  $Y$  is denoted as  $Z$ . Additionally,  $H$  is the entropy computed using the empirical distribution of the random variable. The link strength is set to zero if there is no arc going from  $X$  to  $Y$ . The matrix below displays the percent link strength:

```
R> PLS <- linkStrength(dag = trim.dag, data.df = df, data.dists = dists,
+   method = "ls.pc")
R> rownames(PLS) <- colnames(PLS) <- names(dists)
R> print(PLS, digits = 3)
```

	AR	pneumS	female	livdam	eggs	wormCount	age	adg
AR	0.000	0	0.0000	0	0.0000	0	0.089	0.000
pneumS	0.000	0	0.0000	0	0.0000	0	0.000	0.000
female	0.000	0	0.0000	0	0.0000	0	0.000	0.000
livdam	0.000	0	0.0000	0	0.0405	0	0.000	0.000
eggs	0.000	0	0.0000	0	0.0000	0	0.000	0.128
wormCount	0.282	0	0.0000	0	0.4068	0	0.434	0.376
age	0.000	0	0.0189	0	0.0000	0	0.000	0.000
adg	0.000	0	0.0000	0	0.0000	0	0.390	0.000

#### 4.8. Presentation of the results

The posterior marginals represent Bayesian estimates of the parameters for each node that is associated to the arcs in the DAG. As the variables are coming from different distributions,

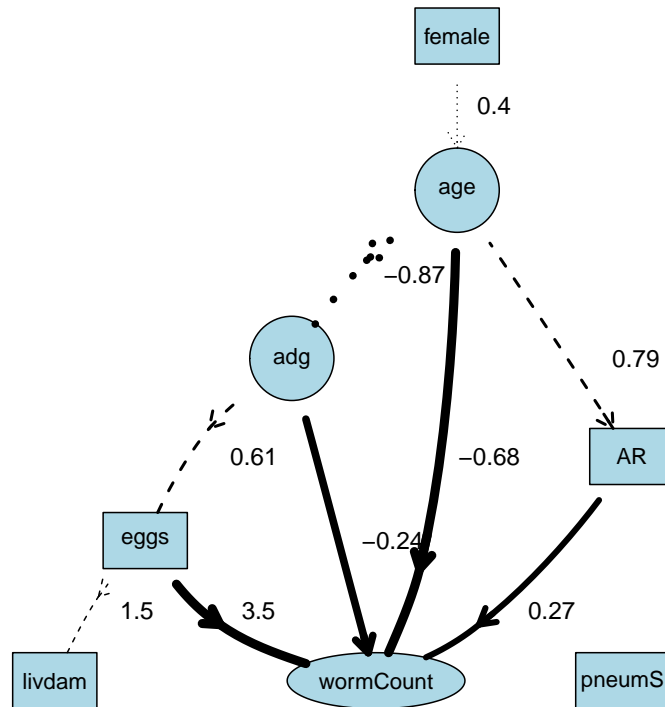


Figure 9: Final ABN model with arc width proportional to link strength. The numbers next to the arcs are the fitted modes. Line styles are according to the interpretation: dashed, solid and dotted for odds ratio, rate ratio and correlation respectively. Box, circle and ellipse nodes represent Bernoulli, Gaussian and Poisson distribution.

the parameters have different interpretations. In Figure 9, the circle nodes are normally distributed, the ellipse node is Poisson distributed and the box nodes are Bernoulli distributed. The posterior marginals represent correlation coefficients for Gaussian nodes, log rate ratios for Poisson nodes, and log odds ratios for binomial nodes. Binomial and Poisson nodes need thus to be exponentiated to get the odds ratios and rate ratios, respectively. Table 3 presents some quantiles of the exponentiated posterior estimates and their interpretation with the PLS values.

## 5. Simulation study

Simulation studies are needed to test models and methods. As an illustration we show here one simulation that illustrates the efficiency of the implemented scoring system. More simulation studies are in the annexes: Appendix A is concerned with structural metrics for DAGs; and Appendix B comprises a quality assessment of the estimation of the regression coefficients.

The parameters which are important from a simulation point of view are the BN dimension, i.e., the number of nodes of the BN, the structure density, i.e., average number of parents per node, and the number of observations. Additionally to these structure-wise metrics, important factors that impact simulations are the arc link strength, the variability of the arc's distributions and the mixture of variables. Indeed, the scores used are only approximately the same over Markov equivalence classes and the discrepancy increases when mixing distributions.

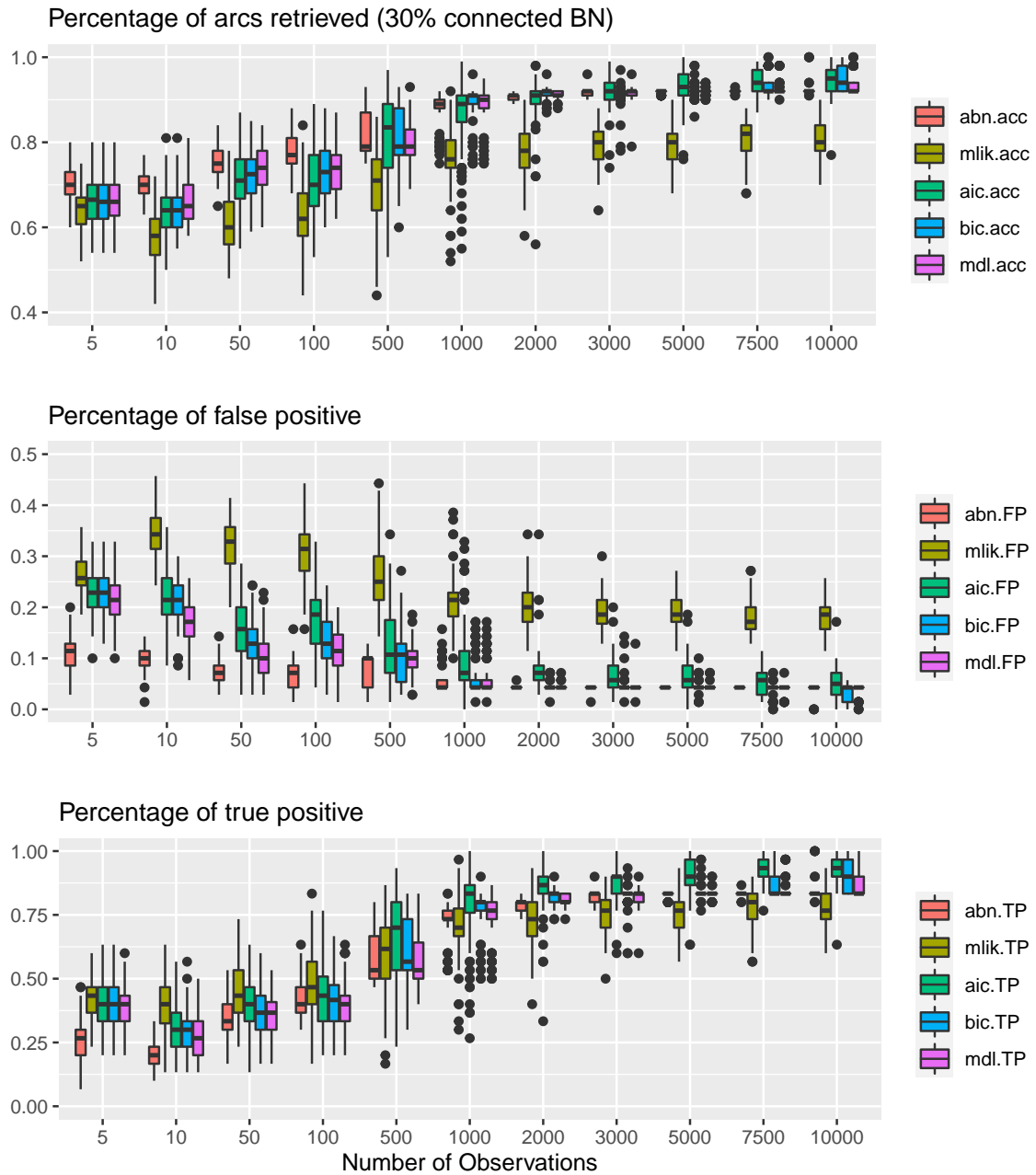


Figure 10: Comparison of score accuracy for marginal posterior likelihood (ABN), maximum likelihood (MLIK), AIC, BIC and MDL for a given BN density in function of the number of observations using accuracy, true positive, false positive and false negative metrics. The DAG has 10 nodes and 29 arcs.

In order to estimate the efficiency of the different scores implemented in the R package **abn**, BNs with a given arc density have been simulated, from which 20 datasets have been generated with different sample sizes. The metrics used to display performance of the scores are the true positive (number of arcs properly retrieved), false positive (selecting an arc when there is not one) and false negative (selecting no arc when there is one). The scores used are marginal

posterior likelihood in a Bayesian regression framework (ABN), maximum likelihood without penalization for complexity (MLIK), AIC, BIC and MDL.

Figure 10 shows for an increasing number of observations  $n = 5, 10, 50, \dots, 10000$  the accuracy of the five scores mentioned above for a given BN density, using boxplots to display the variability of the distribution of the different metrics. As one can see in Figure 10, the maximum likelihood (mlik) is a sub-optimal score for BN learning as expected by [Daly \*et al.\* \(2011\)](#). Interestingly from an applied perspective, the marginal posterior likelihood seems to achieve very promising results with a limited number of observations.

## 6. Summary and discussion

The R package **abn** is a open source implementation of the ABN methodology that allows users to fit Bayesian and MLE ABN models to observational datasets. The R package **abn** contains functions to analyze, select, plot and simulate ABN models. The functions are designed to be usable with limited knowledge of state-of-the-art Bayesian inference methods.

The R package **abn** suffers from several applied and modeling limitations. Some of these limitations are inherent to Bayesian modeling, while some are tentatively tackled in an ABN satellite suite of R packages. These packages are at different development stages, but they are all designed to work synergistically with the R package **abn** and will hopefully be integrated in the longer term.

From an applied perspective, researchers often have to extract a limited number of variables from a large observational dataset. In an ABN context, the exact search algorithm is computationally very sensitive to the number of variables in the network. An ABN analysis targets epidemiological modeling problems where a set of variables are of mutual interest, rather than where only one variable is clearly the outcome (or response) of interest and the remainder are considered predictors. Most of the classical model selection techniques are concerned with this single outcome variable scenario. Driven by these findings, a variable selection approach has been proposed to facilitate ABN analyses; the R package **varrank** ([Kratzer 2022](#); [Kratzer and Furrer 2018](#)) is an implementation of the minimum redundancy maximum relevance model ([Battiti 1994](#)) which performs multi-outcome variable ranking. It can be used prior to the R package **abn** to perform dimensionality reduction and allow the subsequent modeling to focus only on the most important variables. A possible alternative could be to use random forest's variable importance metric which also supports a multi-outcome formulation ([Strobl, Boulesteix, Zeileis, and Hothorn 2007](#)).

A more theoretical limitation of the R package **abn** is that the parameter priors implemented are designed to be non-informative. One can tune some parameters for some parameter priors, but the distributions are fixed. Bayesian model selection algorithms with uninformative priors will asymptotically always select the simpler model, regardless of the data. This is known as Lindley's paradox ([Lindley 1957](#)). A byproduct of more informative priors is the ability to guide the posterior when the likelihood is difficult to estimate. This could be useful when there is (quasi) data separation or in the case of paucity of data. Indeed, while probably suitable for larger datasets, flat priors can quickly turn troublesome when the data is not informative for a parameter of interest ([Kratzer, Furrer, and Pittavino 2019](#)).

One general limitation of the BN modeling approach and the R package **abn** is the strong assumption of data independence (up to clustering) coming from the regression framework

used. Multiple approaches have been proposed to model temporal dynamics, such as dynamic Bayesian networks (DBNs, [Murphy and Russell 2002](#)), temporal nodes Bayesian networks (TNBNs, [Arroyo-Figueroa and Sucar 1999](#)), VAR processes ([Ahelegbey, Billio, and Casarin 2016](#)) and state-space or hidden Markov models ([Le Strat and Carrat 1999](#)).

Two pillars of statistical inference are estimation of effect sizes and quantification of model uncertainty. In a regression context, this is achieved by reporting regression coefficients and their confidence intervals (or sub-optimally using solely the coefficients'  $p$  values). This is done in an ABN analysis when reporting the quantiles of the posterior distribution of the coefficients. However, this is not entirely satisfactory as the uncertainty quantification is performed conditional and marginally on only one model. A superior approach would be to perform Bayesian model averaging, e.g. using MCMC, over many possible structures. The computed MCMC sample could be used to extract arc probabilities of presence or absence. In a broader perspective, it could be possible to compute the probability of any structural query over the MCMC sample. This approach is implemented in the R package **BiDAG** ([Suter, Kuipers, Moffa, and Beerenwinkel 2023](#)) or in the R package **mcmcabn** ([Kratzer 2021; Kratzer et al. 2020](#)). The former is based on an innovative order search, whereas the latter is based on a fully structural formulation. Other available structural MCMC samplers implemented as R packages are **structmcmc** ([Goudie 2020](#)), which includes a Gibbs sampler ([Goudie and Mukherjee 2016](#)) and the (MC)<sup>3</sup> algorithm and **baycn** ([Martin and Fu 2020](#)), which targets causal inference for the direction of the arrows in BNs. However, it is known to be slower in convergence and less efficient with large networks than the R package **BiDAG**. The R package **mcmcabn** has a fully transparent formulation regarding the structural priors used. It eases the interpretation of findings. Moreover, in BN modeling applied to systems epidemiology, due to the limited number of observations that are usually possible from a financial, temporal or logistical perspective, integrating prior knowledge into the inference scheme seems highly desirable.

## 7. Future developments

The case study showed that more exponential distributions should be implemented in order to better grasp the diversity of variables of interest in systems epidemiology. Indeed, zero-inflated variables are poorly represented by the Poisson distribution. A negative binomial would probably be a better alternative. This present paper presents an analysis that takes advantage of bootstrapping to decrease model variance. [Figure 10](#) shows that information theoretic scores perform differently depending on the context. Thus, there could be an opportunity to construct a boosted score dedicated to the ABN methodology. This idea should be addressed in a dedicated simulation study. Finally, it is certainly of interest to extend the R package **abn** to include weakly informative priors instead of flat prior for model's parameters.

## Computational details

The results in this paper were obtained using R version 4.2.2 (2022-11-10) with the package **abn** version 2.7-3 (2023-01-25) on a GNU/Linux 5.15.0-53-generic x86-64 system (Intel Core i7-10510U CPU @ 1.80GHz). The GNU Scientific Library (**GSL**) version is 2.7.1. The **JAGS** version is 4.3.1 available at <https://mcmc-jags.sourceforge.io/>. The **INLA** version is 22.05.07 available at <https://www.R-INLA.org/download-install>.

Calculating the marginal posterior densities rely on C and **GSL** routines which may result in slightly different results under different operating systems when using the same seed. However, the results will be very similar and lead to the same conclusions qualitatively.

## Acknowledgments

G.K. wrote and conceived the manuscript with support from A.C. and R.F. G.K. is author of half of the functions implemented in the R package **abn**. G.K. performed the numerical simulations and contributed to the analysis. G.K. wrote most of the package's documentation. F.I.L. is the original creator and author of half of the functions in the R package **abn**. F.I.L. provided critical feedback and helped to shape the research project. A.C. identified the case study dataset, performed the analysis and contributed to the interpretation of the findings. M.P. contributed to the R package **abn** documentation. R.F. is the PhD supervisor of G.K. and M.P. R.F. probed the R package **abn** and provided useful suggestions that lead to numerous improvements on the package. R.F. provided input on the statistical framework, model implementation and findings reporting. All authors revised the manuscript.

We thank the reviewers and the editorial team, especially Virgilio Gómez-Rubio, for their input and help.

## References

- Ahelegbey DF, Billio M, Casarin R (2016). “Bayesian Graphical Models for Structural Vector Autoregressive Processes.” *Journal of Applied Econometrics*, **31**(2), 357–386. doi:10.2139/ssrn.2198844.
- Arroyo-Figueroa G, Sucar LE (1999). “A Temporal Bayesian Network for Diagnosis and Prediction.” In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 13–20. Morgan Kaufmann Publishers Inc.
- Balov N, Salzman P (2022). **catnet**: *Categorical Bayesian Network Inference*. R package version 1.16.1, URL <https://CRAN.R-project.org/src/contrib/Archive/catnet/>.
- Bang-Jensen J, Gutin GZ (2008). *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag. doi:10.1007/978-1-84800-998-1.
- Battiti R (1994). “Using Mutual Information for Selecting Features in Supervised Neural Net Learning.” *IEEE Transactions on Neural Networks*, **5**(4), 537–550. doi:10.1109/72.298224.
- Belsley DA, Kuh E, Welsch RE (2005). *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, volume 571. John Wiley & Sons. doi:10.1002/0471725153.
- Boerlage B (1992). *Link Strength in Bayesian Networks*. Ph.D. thesis, University of British Columbia.
- Böttcher SG, Dethlefsen C (2003). “**deal**: A Package for Learning Bayesian Networks.” *Journal of Statistical Software*, **8**(20), 1–40. doi:10.18637/jss.v008.i20.

- Bouckaert RR (2008). “Bayesian Network Classifiers in **Weka** for Version 3-5-7.” *Artificial Intelligence Tools*, **11**(3), 369–387. doi:10.1201/b10391-16.
- Breslow NE, Clayton DG (1993). “Approximate Inference in Generalized Linear Mixed Models.” *Journal of the American Statistical Association*, **88**(421), 9–25. doi:10.2307/2290687.
- Comin A, Jeremiasson A, Kratzer G, Keeling L (2019). “Revealing the Structure of the Associations Between Housing System, Facilities, Management and Welfare of Commercial Laying Hens Using Additive Bayesian Networks.” *Preventive Veterinary Medicine*, **164**, 23–32. doi:10.1016/j.prevetmed.2019.01.004.
- Conrady S, Jouffe L (2013). *Introduction to Bayesian Networks & BayesiaLab*. Bayesia SAS.
- Consonni G, La Rocca L (2012). “Objective Bayes Factors for Gaussian Directed Acyclic Graphical Models.” *Scandinavian Journal of Statistics*, **39**(4), 743–756. doi:10.1111/j.1467-9469.2011.00785.x.
- Daly R, Shen Q, Aitken S (2011). “Learning Bayesian Networks: Approaches and Issues.” *The Knowledge Engineering Review*, **26**(2), 99–157. doi:10.1017/s0269888910000251.
- De Jongh M, Druzdzel MJ (2009). “A Comparison of Structural Distance Measures for Causal Bayesian Network Models.” In *Recent Advances in Intelligent Information Systems, Challenging Problems of Science, Computer Science Series*, pp. 443–456.
- Dethlefsen C, Højsgaard S (2005). “A Common Platform for Graphical Models in R: The **gRbase** Package.” *Journal of Statistical Software*, **14**(17), 1–12. doi:10.18637/jss.v014.i17.
- Dohoo IR, Martin W, Stryhn H (2003). *Veterinary Epidemiologic Research*. V413 DOHv. AVC Incorporated, Charlottetown.
- Ebert-Uphoff I (2009). “Tutorial on How to Measure Link Strengths in Discrete Bayesian Networks.” *Technical report*, Georgia Institute of Technology. URL <http://hdl.handle.net/1853/29804>.
- Faraway JJ (2016). *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*. Chapman & Hall/CRC. doi:10.1201/9781315382722.
- Friedman N, Goldszmidt M, Wyner A (1999). “Data Analysis with Bayesian Networks: A Bootstrap Approach.” In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 196–205. Morgan Kaufmann Publishers.
- Furrer R, Kratzer G, Lewis FI (2023). **abn**: *Modelling Multivariate Data with Additive Bayesian Networks*. R package version 2.7-3, URL <https://CRAN.R-project.org/package=abn>.
- Goudie RJB (2020). **structmcmc**: *Structural Inference for Bayesian Networks and Variable Selection*. R package version 1.2, URL <https://github.com/rjbgoudie/structmcmc>.
- Goudie RJB, Mukherjee S (2016). “A Gibbs Sampler for Learning DAGs.” *Journal of Machine Learning Research*, **17**(1), 1–39.



- Hair JF, Anderson RE, Tatham RL, Black WC (1998). *Multivariate Data Analysis*. 5th edition. Prentice Hall, Upper Saddle River.
- Hansen KD, Gentry J, Long L, Gentleman R, Falcon S, Hahne F, Sarkar D (2022). **Rgraphviz**: Provides Plotting Capabilities for R Graph Objects. R package version 2.42.0, URL <https://bioconductor.org/packages/Rgraphviz/>.
- Hartnack S, Odoch T, Kratzer G, Furrer R, Wasteson Y, L'Abée-Lund TM, Skjerve E (2019). “Additive Bayesian Networks for Antimicrobial Resistance and Potential Risk Factors in Non-Typhoidal Salmonella Isolates from Layer Hens in Uganda.” *BMC Veterinary Research*, **15**(1), 212. doi:10.1186/s12917-019-1965-y.
- Heckerman D, Geiger D, Chickering DM (1995). “Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.” *Machine Learning*, **20**(3), 197–243. doi:10.1007/bf00994016.
- Højsgaard S (2012). “Graphical Independence Networks with the **gRain** Package for R.” *Journal of Statistical Software*, **46**(10), 1–26. doi:10.18637/jss.v046.i10.
- Højsgaard S (2021). *CRAN Task View: Graphical Models*. Version 2021-12-27, URL <https://CRAN.R-project.org/view=GraphicalModels>.
- Hornik K, Buchta C, Zeileis A (2009). “Open-Source Machine Learning: R Meets **Weka**.” *Computational Statistics*, **24**(2), 225–232. doi:10.1007/s00180-008-0119-7.
- Kalisch M, Mächler M, Colombo D, Maathuis MH, Bühlmann P (2012). “Causal Inference Using Graphical Models with the R Package **pcalg**.” *Journal of Statistical Software*, **47**(11), 1–26. doi:10.18637/jss.v047.i11.
- Koivisto M, Sood K (2004). “Exact Bayesian Structure Discovery in Bayesian Networks.” *Journal of Machine Learning Research*, **5**, 549–573.
- Kratzer G (2021). **mcmcabn**: A Structural MCMC Sampler for DAGs Learned from Observed Systemic Datasets. R package version 0.5, URL <https://CRAN.R-project.org/package=mcmcabn>.
- Kratzer G (2022). **varrank**: An R Package for Variable Ranking Based on Mutual Information with Applications to Observed Systemic Datasets. R package version 0.5, URL <https://CRAN.R-project.org/package=varrank>.
- Kratzer G, Furrer R (2018). “**varrank**: An R Package for Variable Ranking Based on Mutual Information with Applications to Observed Systemic Datasets.” *arXiv 1804.07134*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1804.07134.
- Kratzer G, Furrer R, Pittavino M (2019). “Comparison between Suitable Priors for Additive Bayesian Networks.” In R Argiento, D Durante, S Wade (eds.), *Bayesian Statistics and New Generation*. Springer-Verlag. doi:10.1007/978-3-030-30611-3\_10.
- Kratzer G, Lewis FI, Willi B, Meli ML, Boretti FS, Hofmann-Lehmann R, Torgerson P, Furrer R, Hartnack S (2020). “Bayesian Network Modeling Applied to Feline Calicivirus Infection among Cats in Switzerland.” *Frontiers in Veterinary Science*, **7**, 73. doi:10.3389/fvets.2020.00073.

- Lauritzen S, Spiegelhalter DJ (1988). “Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems.” *Journal of the Royal Statistical Society B*, **50**, 157–224. doi:10.1111/j.2517-6161.1988.tb01721.x.
- Le Strat Y, Carrat F (1999). “Monitoring Epidemiologic Surveillance Data Using Hidden Markov Models.” *Statistics in Medicine*, **18**(24), 3463–3478. doi:10.1002/(sici)1097-0258(19991230)18:24<3463::aid-sim409>3.0.co;2-i.
- Lewis FI, Ward MP (2013). “Improving Epidemiologic Data Analyses Through Multivariate Regression Modelling.” *Emerging Themes in Epidemiology*, **10**(1), 4. doi:10.1186/1742-7622-10-4.
- Lindley DV (1957). “A Statistical Paradox.” *Biometrika*, **44**(1-2), 187–192. doi:10.1093/biomet/44.1-2.187.
- MacKay DJC (1992). “Bayesian Interpolation.” *Neural Computation*, **4**(3), 415–447. doi:10.1162/neco.1992.4.3.415.
- Madsen AL, Lang M, Kjærulff UB, Jensen F (2003). “The **Hugin** Tool for Learning Bayesian Networks.” In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pp. 594–605. Springer-Verlag.
- Martin EA, Fu AQ (2020). **baycn**: Bayesian Inference for Causal Networks. R package version 1.2.0, URL <https://CRAN.R-project.org/package=baycn>.
- McCormick BJJ (2014). “Frequent Symptomatic or Asymptomatic Infections May Have Long-Term Consequences on Growth and Cognitive Development.” In *Old Herborn University Seminar Monographs*, pp. 23–39. Institute for Microbiology und Biochemistry Herborn, Germany.
- McCormick BJJ, Sanchez-Vazquez MJ, Lewis FI (2013). “Using Bayesian Networks to Explore the Role of Weather as a Potential Determinant of Disease in Pigs.” *Preventive Veterinary Medicine*, **110**(1), 54–63. doi:10.1016/j.prevetmed.2013.02.001.
- Mooij JM (2010). “**libDAI**: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models.” *Journal of Machine Learning Research*, **11**, 2169–2173.
- Murphy K (2001). “The Bayes Net Toolbox for MATLAB.” *Computing Science and Statistics*, **33**(2), 1024–1034.
- Murphy KP, Russell S (2002). “Dynamic Bayesian Networks: Representation, Inference and Learning.” *Technical report*, University of California, Berkeley.
- Pearl J (1985). “Bayesian Networks: A Model for Self-Activated Memory for Evidential Reasoning.” In *Proceedings of the 7th Conference of the Cognitive Science Society*, pp. 15–17. University of California, Irvine,.
- Pearl J (2009). *Causality*. Cambridge University Press.
- Pitman EJG (1936). “Sufficient Statistics and Intrinsic Accuracy.” *Mathematical Proceedings of the Cambridge Philosophical Society*, **32**(4), 567–579. doi:10.1017/s0305004100019307.

- Plummer M (2003). “**JAGS**: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling.” In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*. Technische Universität Wien, Vienna, Austria. URL <https://www.R-project.org/conferences/DSC-2003/Proceedings/Plummer.pdf>.
- Plummer M (2022). **rjags**: *Bayesian Graphical Models Using MCMC*. R package version 4-13, URL <https://CRAN.R-project.org/package=rjags>.
- R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Robinson RW (1973). “Counting Labeled Acyclic Digraphs, New Directions in the Theory of Graphs.” In *Proceedings of the Third Annual Arbor Conference, University of Michigan*, pp. 239–273. Academic Press.
- Robinson RW (1977). “Counting Unlabeled Acyclic Digraphs.” In CHC Little (ed.), *Combinatorial Mathematics V*, pp. 28–43. Springer-Verlag, Berlin.
- Ruchti S, Kratzer G, Furrer R, Hartnack S, Würbel H, Gebhardt-Henrich SG (2019). “Progression and Risk Factors of Pododermatitis in Part-Time Group Housed Rabbit Does in Switzerland.” *Preventive Veterinary Medicine*, **166**, 56–64. doi:10.1016/j.prevetmed.2019.01.013.
- Rue H, Martino S, Chopin N (2009). “Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations.” *Journal of the Royal Statistical Society B*, **71**(2), 319–392. doi:10.1111/j.1467-9868.2008.00700.x.
- Schreiber J (2018). “**Pomegranate**: Fast and Flexible Probabilistic Modeling in Python.” *Journal of Machine Learning Research*, **18**(164), 1–6.
- Scutari M (2010). “Learning Bayesian Networks with the **bnlearn** R Package.” *Journal of Statistical Software*, **35**(3), 1–22. doi:10.18637/jss.v035.i03.
- Scutari M (2018). “Dirichlet Bayesian Network Scores and the Maximum Relative Entropy Principle.” *Behaviormetrika*, **45**(2), 337–362. doi:10.1007/s41237-018-0048-x.
- Spirtes P (2001). “An Anytime Algorithm for Causal Inference.” In TS Richardson, TS Jaakkola (eds.), *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, volume R3 of *Proceedings of Machine Learning Research*, pp. 278–285. PMLR.
- Stehman SV (1997). “Selecting and Interpreting Measures of Thematic Classification Accuracy.” *Remote Sensing of Environment*, **62**(1), 77–89. doi:10.1016/s0034-4257(97)00083-7.
- Strobl C, Boulesteix AL, Zeileis A, Hothorn T (2007). “Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution.” *BMC Bioinformatics*, **8**(1), 25. doi:10.1186/1471-2105-8-25.

- Suter P, Kuipers J, Moffa G, Beerenwinkel N (2023). “Bayesian Structure Learning and Sampling of Bayesian Networks with the R package **BiDAG**.” *Journal of Statistical Software*, **105**(9), 1–31. doi:[10.18637/jss.v105.i09](https://doi.org/10.18637/jss.v105.i09).
- Van Smeden M, De Groot JAH, Moons KGM, Collins GS, Altman DG, Eijkemans MJC, Reitsma JB (2016). “No Rationale for 1 Variable per 10 Events Criterion for Binary Logistic Regression Analysis.” *BMC Medical Research Methodology*, **16**(1), 163. doi:[10.1186/s12874-016-0267-3](https://doi.org/10.1186/s12874-016-0267-3).
- Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.
- Verma T, Pearl J (1988). *Influence Diagrams and D-Separation*. UCLA, Computer Science Department.
- Zou Y, Roos T (2017). “On Model Selection, Bayesian Networks, and the Fisher Information Integral.” *New Generation Computing*, **35**(1), 5–27. doi:[10.1007/s00354-016-0002-y](https://doi.org/10.1007/s00354-016-0002-y).

## A. DAGs simulation: More technical details

In order to simulate DAGs, the function `simulateDag()` generates triangular matrices with user tunable arc density. The node ordering is sampled from the node definition.

In Figure 11, some normalized (i.e., divided by the maximum possible) DAGs structural metrics are displayed: `n.arcs` is the number of arcs, `mb.average` is the average size of the Markov blanket (the set of parents, children and co-parents), `nh.average` is the average number of neighbors, `parent.average` is the average number of parents and `children.average` is the average number of children in the BN. For each level of network complexity, 1000 BN with 40 nodes were simulated. The normalized distribution of the BN metrics computed with `infDag()` is reported. As one can see in Figure 11, the only normalized metric that exhibits non-linear behavior with arc density is the Markov blanket.

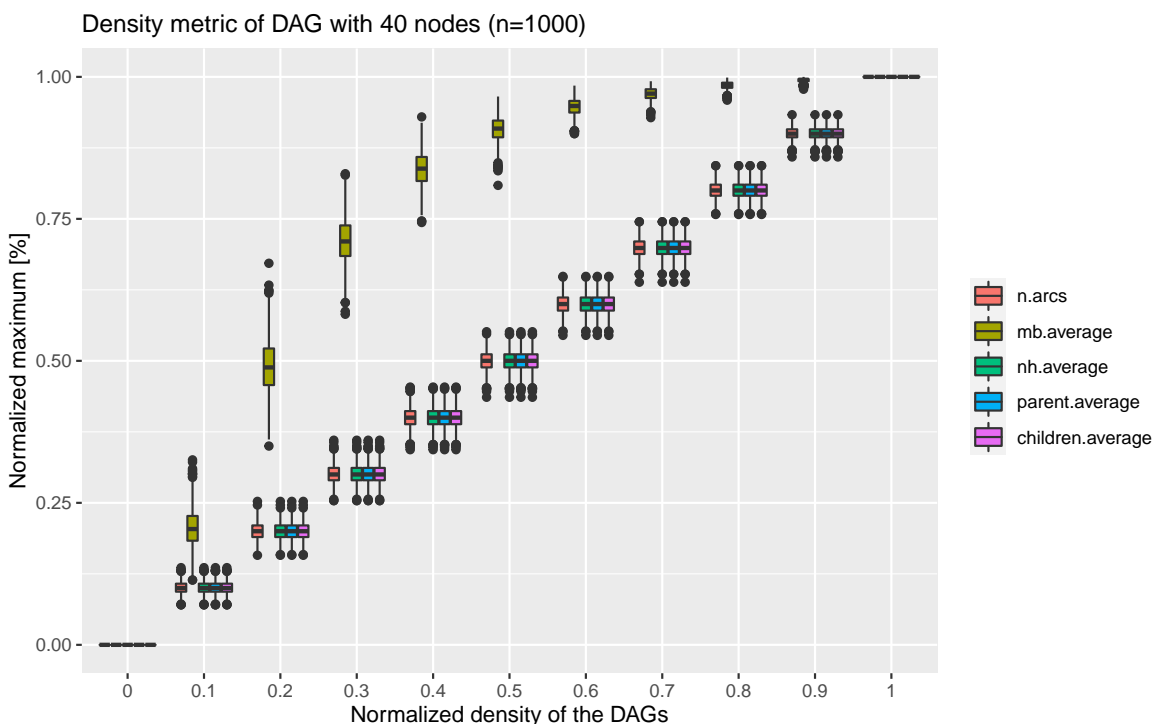


Figure 11: Normalized DAG structural metrics from simulated Bayesian networks of 40 nodes with different arc densities ( $n = 1000$ ).

## B. Regression coefficients estimation: Quality assurance check

The R package `abn` contains two functions to estimate the regression coefficient based on two statistical paradigms: the MLE formulation with an IRLS estimation, and a Bayesian formulation with diffuse priors estimated with `INLA`. Figure 12 shows a quality assurance check of the implementations.

The Bayesian and MLE implementations are compared in Figure 12 to estimate the accuracy of parameters. Two network densities, 20% and 80% of the possible arcs expressed, were simulated 50 times. Then the regression coefficients were computed for different sample sizes

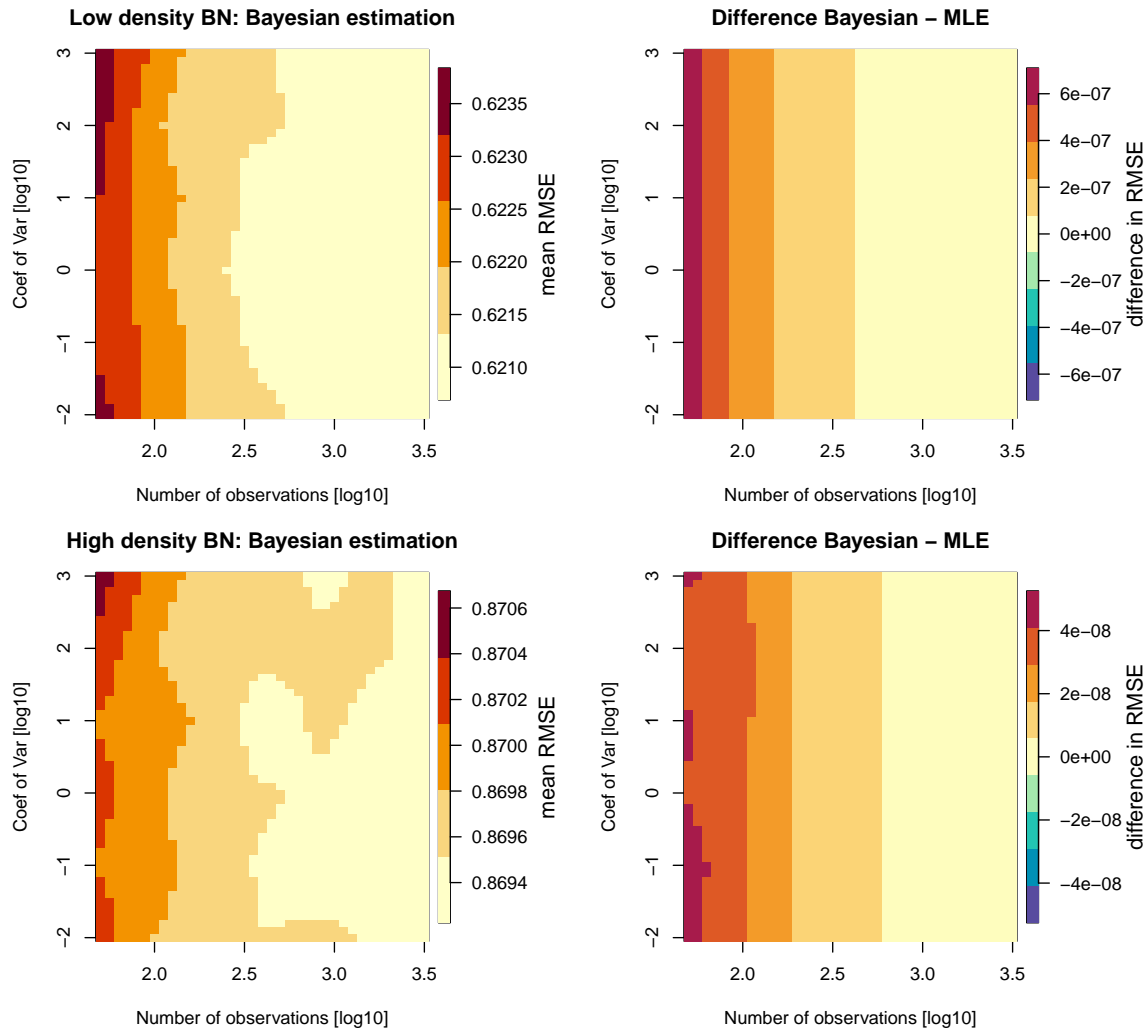


Figure 12: Comparison between Bayesian and MLE implementations to estimate regression coefficients in an ABN framework. The left panels show the mean root mean squared error (mean RMSE) in function of the distribution variability (coefficient of variation) and the sample size for two different network densities. The right panels show the difference between the posterior mode and the MLE. Note the log scales.

and the coefficient of variation of the given node as a proxy for the distribution's variability. As one can see in Figure 12, the error is measured as the mean root mean squared error (mean RMSE) on a log-log scale. Both implementations produce very similar results even if the estimation frameworks are very different.

### C. Parametric bootstrapping: The R code

This section contains the R code needed to perform parametric bootstrapping described by the pseudo-code in Algorithm 1. Starting from the fitted object `marg.f` we extract the marginal densities that will be passed to the **JAGS** model.

```

R> marg.f <- fitAbn(object = mydag, compute.fixed = TRUE)
R> AR.p <- do.call(cbind.data.frame, marg.f$marginals[["AR"]])
R> pneumS.p <- do.call(cbind.data.frame, marg.f$marginals[["pneumS"]])
R> female.p <- do.call(cbind.data.frame, marg.f$marginals[["female"]])
R> livdam.p <- do.call(cbind.data.frame, marg.f$marginals[["livdam"]])
R> eggs.p <- do.call(cbind.data.frame, marg.f$marginals[["eggs"]])
R> wormCount.p <- do.call(cbind.data.frame, marg.f$marginals[["wormCount"]])
R> age.p <- do.call(cbind.data.frame, marg.f$marginals[["age"]][1:2])
R> adg.p <- do.call(cbind.data.frame, marg.f$marginals[["adg"]][1:2])
R> prec.age.p <- marg.f$marginals[["age"]][["age|precision" ]]
R> prec.adg.p <- marg.f$marginals[["adg"]][["adg|precision" ]]

```

We bootstrap 2500 data frames based on one chain with 50000 iterations for adaption and thinning interval of 10.

```

R> nBootstrap <- 2500
R> seeds <- sample(1:100000, nBootstrap)
R> bootout <- foreach(i = 1:length(seeds)) %dopar% {
+   jj <- jags.model(file = "input/model8vPois.bug", data = list(
+     "AR.p" = AR.p, "pneumS.p" = pneumS.p, "female.p" = female.p,
+     "livdam.p" = livdam.p, "eggs.p" = eggs.p, "wormCount.p" = wormCount.p,
+     "age.p" = age.p, "prec.age.p" = prec.age.p, "adg.p" = adg.p,
+     "prec.adg.p" = prec.adg.p),
+     inits = list(".RNG.name" = "base::Mersenne-Twister", ".RNG.seed" =
+     seeds[i]), n.chains = 1, n.adapt = 50000, quiet=TRUE)
+   samp <- coda.samples(jj, c("AR", "pneumS", "female", "livdam", "eggs",
+     "wormCount", "age", "prec.age", "adg", "prec.adg"),
+     n.iter = n.obs * 10, thin = 10)
+   df.boot <- data.frame(samp[[1]][, names(dists)]
+   df.boot[, 1:5] <- lapply(df.boot[, 1:5], function(x) factor(x))
+   mycache <- buildScoreCache(data.df = df.boot, data.dists = dists,
+     dag.banned = banned, max.parents = max.par)
+   mp.dag <- mostProbable(score.cache = mycache)
+   mp.dag$dag
+ }

```

The output consists of the most probable DAG from bootstrapped data frames of the same size as the original. In case more than the most probable DAG is required later, the last three lines need to be read as follows.

```

+   mp.dag <- mostProbable(score.cache = mycache)
+   fabn <- fitAbn(object = mp.dag)
+   list(seeds[i], df.boot, mycache, mp.dat, fabn)
+ }

```

## D. Numerical stability check: Parameter estimation

Once the trimmed DAG is obtained to keep robust structural features, we can extract the marginal posterior densities. Indeed, a BN has a qualitative part (the structure) and a quantitative part (parameters estimates). They are both equally important for interpreting and reporting the findings. As the estimation is based on the Laplace approximation and the R package **abn** does not rely on conjugate priors, a numerical check is performed. Figure 8 shows the marginal posterior densities with and without correction for clustering. The critical assumption is that the data contains sufficient information to accurately estimate the density of individual parameters of the model. This is a stronger requirement than simply being able to estimate an overall goodness of fit metric. It could happen that **INLA** or the internal C code cannot estimate the densities with enough numerical stability. Many user-tunable parameters can be supplied to `fitAbn()` by providing a list to the `control` argument.

The following code exemplifies the extraction of the marginal posterior densities and the construction of Figure 8 based on the fitted objects `marg.f` and `marg.f.grouped`.

```
R> par(mfrow = c(5, 4), mar = c(2, 2, 1.5, 1))
R> for(i in 1:length(marg.f$marginals)){
+   cur.node <- marg.f$marginals[i][[1]]
+   cur.node.gr <- marg.f.grouped$marginals[i][[1]]
+   nom1 <- names(marg.f$marginals)[i]
+   for(j in 1:length(cur.node) ) {
+     nom2 <- names(cur.node)[j]
+     cur.param <- cur.node[[j]]
+     cur.param.gr <- cur.node.gr[[j]]
+     if(grepl("precision", nom2)){
+       cur.param[, 1] <- 1/cur.param[, 1]
+       cur.param.gr[, 1] <- 1/cur.param.gr[, 1]
+       nom2 <- gsub("precision", "variance", nom2)
+     }
+     plot(cur.param, type = "l", main = paste(nom1, ":", nom2), cex = 0.7,
+          xlim = range(cur.param[, 1], cur.param.gr[, 1]),
+          ylim = c(0, max(cur.param[, 2], cur.param.gr[, 2])))
+     lines(cur.param.gr, type = "l", col = "blue")
+   }
+ }
```

Marginal posterior densities that are oscillating or strongly asymmetric are an indication of numerical instabilities. A simple additional test is the area under the density, as shown below.

```
R> AUC <- function(x)
+   sum(diff(x[, 1]) * (x[-1, 2] + x[-length(x[, 1]), 2]) / 2)
R> aucadg <- lapply(marg.f.grouped$marginals, function(x) lapply(x, AUC))
R> summary(unlist(aucadg))
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
	0.9999	1.0000	1.0000	1.0000	1.0000	1.0000



**Affiliation:**

Reinhard Furrer

Department of Mathematics *and* Institute of Computational Science

University of Zurich

Winterthurerstrasse 190

8057 Zurich, Switzerland

E-mail: [reinhard.furrer@math.uzh.ch](mailto:reinhard.furrer@math.uzh.ch)

URL: <http://r-bayesian-networks.org/>