Journal of Statistical Software

November 2019, Volume 91, Issue 11. *doi: 10.18637/jss.v091.i11*

Learning Large-Scale Bayesian Networks with the sparsebn Package

Bryon Aragam
University of Chicago

Jiaying Gu
University of California, Los Angeles

Qing Zhou
University of California, Los Angeles

Abstract

Learning graphical models from data is an important problem with wide applications, ranging from genomics to the social sciences. Nowadays datasets often have upwards of thousands – sometimes tens or hundreds of thousands – of variables and far fewer samples. To meet this challenge, we have developed a new R package called **sparsebn** for learning the structure of large, sparse graphical models with a focus on Bayesian networks. While there are many existing software packages for this task, this package focuses on the unique setting of learning large networks from high-dimensional data, possibly with interventions. As such, the methods provided place a premium on scalability and consistency in a high-dimensional setting. Furthermore, in the presence of interventions, the methods implemented here achieve the goal of learning a causal network from data. Additionally, the **sparsebn** package is fully compatible with existing software packages for network analysis.

Keywords: Bayesian networks, causal networks, graphical models, machine learning, structural equation modeling, multi-logit regression, experimental data.

1. Introduction

Graphical models are a popular tool in machine learning and statistics, and have been used in a variety of applications including genetics (Gao and Cui 2015; Isci, Dogan, Ozturk, and Otu 2014), computational biology (Jones, Buchan, Cozzetto, and Pontil 2012), oncology (Chen *et al.* 2015), medicine and health care (Nicholson, Cozman, Velikova, Van Scheltinga, Lucas, and Spaanderman 2014), logistics (Garvey, Carnovale, and Yeniyurt 2015), finance (Sanford and Moosa 2012), and even software testing (Dejaeger, Verbraken, and Baesens 2013). The widespread growth of high-dimensional biological data in particular has spurred a renewed interest in the use of graphical models to aid in the discovery of novel biological mechanisms (Bühlmann, Kalisch, and Meier 2014). While the past decade has witnessed tremendous developments towards understanding undirected graphical models (Meinshausen and Bühlmann 2006; Ravikumar, Wainwright, and Lafferty 2010; Yang, Ravikumar, Allen, and Liu 2015), there has been less progress towards understanding directed graphical models – also known as Bayesian networks (BNs) or structural equation models (SEM) – for high-dimensional data with $p \gg n$. A BN is represented by a directed acyclic graph (DAG), whose structure contains a richer and different set of conditional independence relations than an undirected graph. Moreover, DAGs are commonly used in causal inference where the direction of an edge encodes causality. Consequently, there have been continuing efforts in structure learning of directed graphs from data.

Unlike their undirected counterparts, however, the structure learning problem for directed graphical models is complicated by the nonconvexity, nonsmoothness, and nonidentifiability of the underlying statistical problem. These issues have no doubt slowed progress towards fast, scalable algorithms for learning in the presence of thousands – let alone tens of thousands – of variables. Despite progress at the theoretical (Van de Geer and Bühlmann 2013; Aragam, Amini, and Zhou 2015) and computational level (Schmidt, Niculescu-Mizil, and Murphy 2007; Xiang and Kim 2013; Fu and Zhou 2013), there is still a lack of user-friendly software for putting these modern tools into the hands of practitioners.

To bridge this gap we have developed **sparsebn**, a new R (R Core Team 2019) package for structure learning and parameter estimation of large-scale Bayesian networks from high-dimensional data. When experimental data are available, an estimated DAG from this package has a natural causal interpretation. While there are many R packages for learning Bayesian networks (Section 2.4), none that we are aware of are specifically tailored to high-dimensional data with experimental interventions. The **sparsebn** package has been developed from the ground up using recent developments in structure learning (Fu and Zhou 2013; Aragam and Zhou 2015; Gu, Fu, and Zhou 2018) and statistical optimization (Friedman, Hastie, Höfling, and Tibshirani 2007; Friedman, Hastie, and Tibshirani 2010; Mazumder, Friedman, and Hastie 2011). In addition to methods for learning Bayesian networks, this package also includes procedures for learning undirected graphs, fitting structural equation models, and is compatible with existing packages in R. All of the code for this package is open-source and available through the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=sparsebn`.

To briefly illustrate the use of **sparsebn**, the code below learns the structure of the pathfinder network (Heckerman, Horvitz, and Nathwani 1992):

```
R> library("sparsebn")
R> data("pathfinder")
R> data <- sparsebnData(pathfinder$data, type = "continuous")
R> dags <- estimate.dag(data)
R> plotDAG(dags)
```

This code estimates a *solution path* with 16 total estimates (see Section 3.2) and takes approximately one second to run. The first four estimated networks with an increasing number of edges are shown in Figure 1. This example is explored in more detail in Section 6.2.

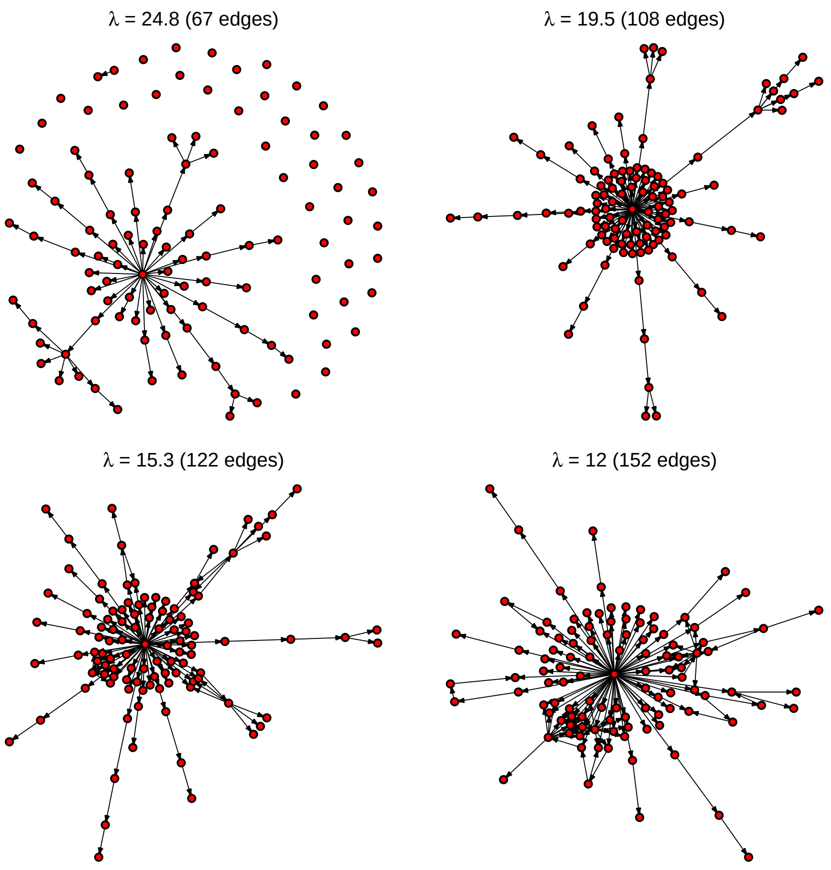


Figure 1: Example output from learning the pathfinder network. To save space, only the first four nontrivial estimates are shown here out of the 16 total estimates in the full solution path.

2. Learning Bayesian networks from data

We begin by reviewing the necessary background and definitions, and then discuss the existing literature and methods.

2.1. Background

The basic model we work with is a p-dimensional random vector $X = (X_1, \dots, X_p)$ with joint distribution $\mathsf{P}(X_1, \dots, X_p)$. Bayesian networks are directed graphical models whose edges encode conditional independence constraints implied by the joint distribution of X. For continuous data, we assume that X follows a multivariate Gaussian distribution; for discrete data we assume each X_j is a factor with r_j levels. We do not consider so-called hybrid Bayesian networks that allow for graphs with both continuous and discrete nodes, although this is an interesting future direction for this package.

The general theory of Bayesian networks is quite intricate, and we will make no attempt to cover it in detail here. The interested reader is referred to one of the many excellent textbooks on this subject: Koller and Friedman (2009); Spirtes, Glymour, and Scheines (2000); Lauritzen (1996).

Formally, a Bayesian network is defined as a directed acyclic graph $G = (V, E)$ that satisfies the following factorization condition with respect to the joint distribution of X:

$$\mathsf{P}(X_1, \ldots, X_p) = \prod_{j=1}^{p} \mathsf{P}(X_j \mid \text{pa}(X_j), \theta_j).$$

Here, $\text{pa}(X_j) = \{X_i : X_i \to X_j \in E\}$ is the parent set of X_j and θ_j encodes the parameters that define the conditional probability distribution (CPD) for X_j.

Traditional methods for learning Bayesian networks start with this definition and develop algorithms from a graph-theoretic perspective (Spirtes and Glymour 1991). This approach comes with restrictive assumptions such as strong faithfulness (Uhler, Raskutti, Bühlmann, and Yu 2013; Zhang and Spirtes 2002), which hinders their use in practice. To motivate our work, we adopt a more general approach to Bayesian networks via structural equation models. In this approach, we start by directly modeling each CPD $\mathsf{P}(X_j \mid \text{pa}(X_j), \theta_j)$ via a generalized linear model. We will consider two special cases: Gaussian CPDs for continuous data and multi-logit CPDs for discrete data. This approach also fits naturally our framework for learning causal networks discussed in Section 2.2.

Continuous data

Suppose that for each $j = 1, \ldots, p$ there exists $\beta_j = (\beta_{1j}, \ldots, \beta_{pj}) \in \mathbb{R}^p$ such that

$$X_j = \beta_j^\top X + \varepsilon_j, \tag{1}$$

where $\beta_{jj} = 0$ (to avoid trivialities) and $\varepsilon_j \sim \mathcal{N}(0, \omega_j^2)$. By writing $B = [\beta_1 \mid \cdots \mid \beta_p] \in \mathbb{R}^{p \times p}$ and $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_p) \in \mathbb{R}^p$, we can rewrite Equation 1 as a matrix equation:

$$X = B^\top X + \varepsilon. \tag{2}$$

The model (2) is called a *structural equation model* for X. The matrix B defines the weighted adjacency matrix of a directed graph which, when acyclic, is also a BN for X. Note that to estimate B from data it is not enough to simply regress X_j onto the rest of variables – this leads to the so-called *neighbourhood regression* estimator of the Gaussian graphical model introduced in Meinshausen and Bühlmann (2006). The problem with this approach is that there is no guarantee that the resulting adjacency matrix will be acyclic. In order to produce a BN, we must constrain B to be acyclic, which couples the parameters of each CPD to one another and induces a nonconvex constraint during the learning phase. For this reason, learning directed graphs is substantially more difficult than learning undirected graphs.

By writing $\Omega = \mathsf{COV}(\varepsilon)$, we see that the parameters (B, Ω) specify a unique normal distribution $\mathcal{N}(0, \Sigma)$ for X. In fact, a little algebra on Equation 2 shows that

$$\Sigma = (I - B)^{-\top} \Omega (I - B)^{-1}. \tag{3}$$

This gives a way to compute Σ from (B, Ω), and suggests a way to estimate the covariance matrix by setting $\widehat{\Sigma} = (I - \widehat{B})^{-\top} \widehat{\Omega} (I - \widehat{B})^{-1}$. By taking $\Gamma = \Sigma^{-1}$, we obtain an alternative estimator of the Gaussian graphical model, given by

$$\widehat{\Gamma} = (I - \widehat{B}) \widehat{\Omega}^{-1} (I - \widehat{B})^\top. \tag{4}$$

This idea was first explored by Rütimann and Bühlmann (2009) using the PC algorithm (Section 2.3), and a similar idea using our methods is implemented in **sparsebn**. In many situations, the DAG representation (B, Ω) encodes more conditional independence relations than the inverse covariance matrix Γ, which is one of the motivations for learning DAGs from observational data.

Discrete data

In a discrete Bayesian network, each X_j is a finite random variable with r_j states. Instead of a traditional product multinomial model, **sparsebn** uses a multi-logit model for discrete data. One of the advantages of this approach is a significant reduction in the number of parameters, and recent work suggests that this model serves as a good approximation to the product multinomial model (Gu *et al.* 2018). Here, we briefly introduce the multi-logit model, and describe how it is used for structure learning.

We use a standard form of the multi-logit model in which each variable X_j is encoded by $d_j = r_j - 1$ dummy variables (Dobson and Barnett 2008). More specifically, given a reference category (which may be arbitrary), the r_j possible values of X_j are encoded by a vector of dummy variables $\mathbf{z}_j = (z_{jk}, k = 1, \ldots, d_j) \in \{0,1\}^{d_j}$. Let $I(\cdot)$ denote the usual indicator function, so that e.g., $I(X_j = k) = 1$ if $X_j = k$ and $I(X_j = k) = 0$ otherwise. If we choose the last category as the reference, then $z_{jk} = I(X_j = k)$ for $k = 1, \ldots, d_j$ so that the reference category is coded as $\mathbf{z}_j = \mathbf{0}$.

Under this parametrization, the conditional distributions take the form

$$\mathsf{P}(X_j = u \mid \mathrm{pa}(X_j)) = \frac{\exp\left(\beta_{0uj} + \sum_{i \in \mathrm{pa}(X_j)} \mathbf{z}_i^\top \boldsymbol{\beta}_{iuj}\right)}{\sum_{m=1}^{r_j} \exp\left(\beta_{0uj} + \sum_{i \in \mathrm{pa}(X_j)} \mathbf{z}_i^\top \boldsymbol{\beta}_{imj}\right)}, \quad u = 1, \ldots, r_j, \tag{5}$$

where $\boldsymbol{\beta}_{iuj} \in \mathbb{R}^{d_i}$ is the coefficient vector for variable X_i to predict the uth level of X_j with intercept β_{0uj}. In this model, if $X_i \notin \mathrm{pa}(X_j)$ then we have equivalently $\boldsymbol{\beta}_{iuj} = \mathbf{0}$ for all u. Let $\boldsymbol{\beta}_{ij} = (\boldsymbol{\beta}_{i1j}, \ldots, \boldsymbol{\beta}_{ir_jj}) \in \mathbb{R}^{d_i r_j}$ be the coefficient vector for edge i to j. Therefore, by estimating $\boldsymbol{\beta}_{ij}$, we can infer the structure of a network. If $\boldsymbol{\beta}_{ij} = \mathbf{0}$, there does not exist an edge from X_i to X_j. For more details on this model, see Gu *et al.* (2018).

2.2. Causal DAG learning from interventions

DAGs are a popular model for causal networks, particularly when combined with experimental interventions in addition to observational data (Pearl 2000). Cooper and Yoo (1999); Meganck, Leray, and Manderick (2006); Ellis and Wong (2008) proposed methods to learn causal networks with a mixture of observational and experimental data, and Peér, Regev, Elidan, and Friedman (2001); Pournara and Wernisch (2004) inferred gene networks with perturbed expression data. Each of the methods in **sparsebn** can take experimental data as input in order to learn the precise causal relationships in a system.

The following simple example illustrates how interventions can be used for learning causal relations: Assume the true causal graph is $\mathcal{G}^* : X_1 \to X_2$, i.e., X_1 is a direct cause of X_2. This is observationally equivalent to the graph $\mathcal{G} : X_1 \leftarrow X_2$ – we cannot distinguish between these two graphs using observational data alone. Instead, by manipulating X_2 experimentally and fixing its value, we can "cut off" the edge from X_1 to X_2 since the value of X_2 would no longer be associated with (i.e., is independent of) X_1. This is an example of an *experimental* *intervention.* In doing so, the joint distribution factors as $\mathsf{P}(X_1)\mathsf{P}(X_2)$. Instead, if we manipulated X_1, the relation between X_1 and X_2 would stay the same, and the joint distribution factors as $\mathsf{P}(X_1)\mathsf{P}(X_2 \mid X_1)$. By exploiting these interventions, it is possible to uncover the true causal structure in a physical system.

To see how this can be applied in a statistical setting, we now show how a general form of the density function for experimental data can be derived from the pure observational joint density function. Let $\mathcal{M} \subset \{1, \ldots, p\}$ be the set of variables under intervention, so the joint probability decomposes as

$$\mathsf{P}(X_1, \ldots, X_p) = \prod_{i \notin \mathcal{M}} \mathsf{P}(X_i \mid \mathrm{pa}(X_i)) \prod_{i \in \mathcal{M}} \mathsf{P}(X_i \mid \bullet), \tag{6}$$

where $\mathsf{P}(X_i \mid \bullet)$ is the marginal distribution of X_i from which experimental samples are drawn. Thus, experimental data sets generated from the true DAG $\mathcal{G}$ can be considered as data sets generated from a DAG $\mathcal{G}'$, where $\mathcal{G}'$ is obtained by removing all directed edges in $\mathcal{G}$ pointing to the variables under intervention. Furthermore, we can see that when $\mathcal{M} = \emptyset$, Equation 6 is the density function for observational data. For more details regarding causal DAG learning, please refer to Pearl (2000) and the references therein.

2.3. Previous work

The various algorithms in the literature for structure learning of Bayesian networks fall into three main categories: constraint-based methods, score-based methods and hybrid methods.

Constraint-based methods

Constraint-based methods rely on repeated conditional independence tests in order to learn the structure of a network. The main idea is to determine which edges cannot exist in a DAG using statistical tests of independence, a procedure which is justified whenever the so-called *faithfulness* assumption holds. These algorithms first use independence tests to learn the skeleton of the network, and then orient v-structures along with the rest of the edges. Because of the existence of Markov equivalent DAGs, the direction of some edges may not be decided (for details, see e.g., Koller and Friedman 2009). The PC algorithm proposed by Spirtes and Glymour (1991) is a well-known example of this kind of method. Another example is the Fast Causal Inference (FCI) algorithm (Spirtes *et al.* 2000; Colombo, Maathuis, Kalisch, and Richardson 2012), which allows for latent variables in the network. The output of these algorithms is a partially directed graph, which means that there may be some undirected edges in the estimated graph. While the PC algorithm is a powerful method to learn Bayesian networks in low-dimensions with n very large, the performance of the PC algorithm is less competitive in high-dimensions compared to recent score-based methods (Aragam and Zhou 2015).

Score-based methods

Score-based methods rely on scoring functions such as the log-likelihood or some other loss function. The goal of these algorithms is to find a DAG that optimizes a given scoring function. Some popular scoring functions include several Bayesian Dirichlet metrics (Buntine 1991; Cooper and Herskovits 1992; Heckerman, Geiger, and Chickering 1995), Bayesian information criterion (Chickering and Heckerman 1997), minimum description length (Bouckaert 1993;

Suzuki 1993; Lam and Bacchus 1994), and entropy (Herskovits and Cooper 1990). One of the classic score-based methods is the *greedy hill climbing* (HC) algorithm (Russell and Norvig 1995). This algorithm is fast but tends to predict too many edges in high-dimensional settings. For discrete networks, the K2 algorithm (Cooper and Herskovits 1992) is another popular method, however, this method requires prior knowledge about the ordering of the network which is often unavailable in applications. There are also Monte Carlo methods (Ellis and Wong 2008; Zhou 2011; Niinimäki, Parviainen, and Koivisto 2016), which are quite accurate but also computationally demanding. This limits Monte Carlo methods to smaller networks with only tens of nodes. Each of the learning algorithms implemented in **sparsebn** is a score-based method, and as such this package represents an attempt to resolve many of the computational and statistical issues cited here with respect to these methods.

Hybrid methods

Finally, there are hybrid methods which combine constraint-based and score-based methods. Hybrid methods first prune the search space by using a constraint-based search, and then learn an optimal DAG structure via score-based search (Tsamardinos, Brown, and Aliferis 2006; Perrier, Imoto, and Miyano 2008; Gámez, Mateo, and Puerta 2011). The max-min hill-climbing (MMHC) algorithm proposed by Tsamardinos *et al.* (2006) is a powerful method of this kind.

2.4. Existing R packages for structure learning

There are several existing R packages for learning and manipulating Bayesian networks. **bnlearn** is a thorough and actively maintained package that implements a wide variety of classical approaches such as HC and MMHC (Scutari 2010, 2017). **pcalg** focuses on the causal interpretation of Bayesian networks, and implements the PC and FCI algorithms along with methods for inferring causal effects (Kalisch, Mächler, Colombo, Maathuis, and Bühlmann 2012). Other packages include **deal** for mixed data (Boettcher and Dethlefsen 2003) and **gRain** (Højsgaard 2012) for exact and approximate computations. For structural equation models in particular, **lavaan** is a modern R package that implements many of the standard fitting procedures for SEM (Rosseel 2012). There are also many software packages available on platforms besides R (for a more complete list see Murphy 2014) such as **BEANDisco** (Niinimäki *et al.* 2016, `https://www.cs.helsinki.fi/u/tzniinim/BEANDisco/`), **AMIDST** (Masegosa *et al.* 2017, `http://www.amidsttoolbox.com/`), and **GOBNILP** (Cussens, Haws, and Studený 2017, `https://www.cs.york.ac.uk/aig/sw/gobnilp/`).

Motivated by applications to computational biology and machine learning, **sparsebn** was designed for the following types of applications:

- Datasets with several thousand variables, which arise in computational biology and machine learning,
- High-dimensional data with $p \gg n$, which is common in genomics applications with high-throughput datasets, such as gene expression data that have $p \sim 20,000$ and $n \sim 100$,
- Experimental data with interventions, which is also common in genomics applications.

Unfortunately, the aforementioned packages either do not scale to handle these types of problems, or cannot process high-dimensional data at all. By contrast, the **sparsebn** package was specifically designed to fill this gap, with an orientation towards large, high-dimensional, experimental data. In order to achieve this, the methods contained in this package rely on a combination of novel methodology and algorithms in order to scale to larger and larger datasets. Of course, these methods also gracefully degrade to handle simpler settings with observational and/or low-dimensional data. Moreover, whenever possible, cross-compatibility with the above mentioned packages has been provided (Section 4.5).

3. Learning with sparse regularization

To learn a Bayesian network from data, we use a score-based approach based on regularized maximum likelihood estimation that allows for the incorporation of experimental data. In this section we discuss these details as well as the block coordinate descent algorithm used to approximate the resulting optimization problem.

3.1. Regularized maximum likelihood

Suppose $\mathbf{X} \in \mathbb{R}^{n\times p}$ is a matrix of observations, and let ℓ denote the negative log-likelihood and ρ_λ be some regularizer. For example, ρ_λ may be the ℓ_1 penalty (Tibshirani 1996), the group norm penalty (Yuan and Lin 2006), or a nonconvex penalty such as the smoothly clipped absolute deviation (SCAD, Fan and Li 2001) or minimax concave penalty (MCP, Zhang 2010). Furthermore, we assume that $\mathbf{X}$ does not contain any missing values. If the data contains missing values, these should be imputed first.

We consider the following program:

$$\min_{B\in\mathbb{D}} \ell(B;\mathbf{X}) + \rho_\lambda(B), \tag{7}$$

where $\mathbb{D} \subset \mathbb{R}^{p\times p}$ is the set of weighted adjacency matrices that represent directed graphs without cycles. The resulting problem (7) is a nonconvex program, where the nonconvexity arises from (potentially) all three terms: The constraint $\mathbb{D}$, the loss function ℓ, and the regularizer ρ_λ.

For continuous data, we use a Gaussian likelihood derived from Equation 2 combined with ℓ_1 or MCP regularization, and for discrete data we use a multi-logit model as in Equation 5 combined with a group lasso penalty. When some data are generated under experimental intervention, we can derive the form of $\ell(B;\mathbf{X})$ using the strategy discussed in Section 2.2: When X_i is under intervention, its marginal distribution is known and hence can be considered constant in Equation 6. Let $\mathcal{I}_j$ be the set of row indices of the data matrix $\mathbf{X}$ where node X_j is under intervention, and $\mathcal{O}_j = \{1,\ldots,n\} - \mathcal{I}_j$ be the collection of observations for which X_j is not under intervention. Then, according to Equation 6, the negative log-likelihood factorizes as

$$\ell(B;\mathbf{X}) = -\sum_{j=1}^{p}\sum_{h\in\mathcal{O}_j} \log f_{\beta_j}(\mathbf{x}_{hj}\,|\,\mathrm{pa}(\mathbf{x}_{hj})), \tag{8}$$

where $B = [\,\beta_1\,|\cdots|\,\beta_p\,]$, f_{β_j} is the conditional density for the jth node, and $\mathbf{x}_{hj}$ is the value of node X_j at the hth data point. Note that multiple nodes may be intervened for a particular data point. By incorporating experimental interventions in this way, we are able to orient the edges in a causal Bayesian network and thus distinguish between equivalent DAGs. A similar strategy from a Bayesian perspective was first adopted in Cooper and Yoo (1999). Evidently, one advantage of this framework is its universal applicability to different data types and various likelihood models.

Since the program (7) is nonconvex it is generally regarded as infeasible to find an exact global minimizer of this problem. Indeed, score-based structure learning is known to be NP-hard (Chickering, Heckerman, and Meek 2004). Instead, we seek local minimizers of (7) through an optimization scheme based on block coordinate descent. The method is based on the following observation: The difficulty in solving (7) lies in enforcing the constraint $B \in \mathbb{D}$, which is a highly nonconvex and singular parameter constraint. Instead, if we consider (7) *one* edge at a time, the problem simplifies considerably. This is the heuristic exploited by many score-based methods (most notably greedy hill climbing). Unlike conventional methods, however, Fu and Zhou (2013) propose a *block-cyclic* strategy and show that it outperforms existing approaches based on greedy updates. This general observation has been exploited to construct the family of fast algorithms implemented in **sparsebn**.

3.2. Algorithm details

Recalling that $B = (\beta_{ij})$, the high-level idea behind the method is the following:

1. Repeat outer loop until stopping criterion met:
2. *Outer loop.* For each pair (j, k), $j \neq k$:
   (a) Minimize (7) with respect to (β_{kj}, β_{jk}), holding all other parameters fixed;
   (b) If the edge $k \to j$ (resp. $j \to k$) induces a cycle in the graph, set $\beta_{kj} \leftarrow 0$ (resp. $\beta_{jk} \leftarrow 0$) and then update β_{jk} (resp. β_{kj});
   (c) Repeat inner loop until convergence:
      3. *Inner loop.* Fix the edge set E from the outer loop and minimize (7) by cycling through the edge weights β_{kj} for $(k, j) \in E$.

Since the program (7) depends on the unknown regularization parameter λ, this value must be supplied in advance to the algorithm. In practice, we wish to solve (7) for several values of the regularization parameter, so instead of returning a single DAG estimate, the output of this meta-algorithm is a *solution path* (also called a *regularization path*, Friedman *et al.* 2010): A sequence of estimates $\{\widehat{B}(\lambda_{\max}), \widehat{B}(\lambda_1), \ldots, \widehat{B}(\lambda_{\min})\}$ for a pre-determined grid of values $\lambda_{\max} > \lambda_1 > \cdots > \lambda_{\min}$. This is standard practice in the literature on coordinate descent (Friedman *et al.* 2007; Wu and Lange 2008), and is similar to the well-known graphical lasso for undirected graphs (Friedman, Hastie, and Tibshirani 2008).

As λ decreases, there is less regularization, hence the resulting estimates $\widehat{B}(\lambda_m)$ become more dense (i.e., contain more edges). Since our focus is on sparse graphs, in practice we use this fact to terminate the algorithm early if the solution path becomes too dense, i.e., if the number of edges in $\widehat{B}(\lambda_m)$ exceeds some user-defined threshold. The default values used in **sparsebn** are $10p$ edges for continuous data and $3p$ edges for discrete data. The smaller threshold for discrete data is due to the higher computational complexity of the underlying algorithm as compared with the continuous case.

The detailed implementation of the algorithms uses several tricks from the literature on coordinate descent in order to speed up the algorithm:

- *Warm starts.* Given the previous estimate $\widehat{B}(\lambda_{m-1})$ in the solution path, we use $\widehat{B}(\lambda_{m-1})$ as the initial guess for the next iterate $\widehat{B}(\lambda_m)$. Furthermore, it is always possible to choose $\lambda_{\max}$ so that $\widehat{B}(\lambda_{\max}) = \mathbf{0}$ (i.e., the zero matrix), which is the default implementation used by **sparsebn**.
- *Active set iterations.* In the inner loop above, the algorithm only updates the nonzero parameters by solving at most p penalized regression or multi-logit regression problems. These subproblems are computationally tractable and can be solved efficiently, which yields significant performance improvements for large graphs.
- *Block updates.* Instead of updating each β_{jk} one at a time, the algorithms update each parameter as a block $\{\beta_{jk}, \beta_{kj}\}$. This is justified by the acyclicity assumption: If $\beta_{jk} \neq 0$ and $\beta_{kj} \neq 0$, then the acyclic constraint is violated, and this fact is exploited to update both parameters simultaneously.
- *Sparse data structures.* Internally, everything is stored using sparse data structures for representing directed acyclic graphs. This saves memory and speeds up the computation of each update.

Compared with existing packages for structure learning, the main novelties of the present methods are (1) The use of ℓ_1 and MCP regularization, (2) Block-cyclic (as opposed to greedy, one-at-a-time) updates, and (3) The use of warm starts in computing the solution path. Further details can be found in Fu and Zhou (2013) and Aragam and Zhou (2015).

3.3. Parameter estimation

After learning the structure of a Bayesian network, often it is of interest to then estimate the parameters of the local conditional distributions associated with the learned structure. For causal DAGs, these parameters determine the causal effect sizes between the parents and their children.

For Gaussian data, this is straightforward by regressing each node onto its parents, using (unpenalized) least squares regression. Note that this requires that the maximum number of parents is at most n, which is another motivation for leveraging sparsity in our algorithms. This produces a *weighted* adjacency matrix $\widehat{B} = (\widehat{\beta}_{ij})$ (or more specifically, a solution path of adjacency matrices). Given these weights, we can estimate the conditional variance of each node given its parents by:

$$\widehat{\omega}_j^2 := \mathsf{VAR}(\mathbf{x}_j - \mathbf{X}\widehat{\beta}_j).$$

This yields a variance matrix $\widehat{\Omega} = \operatorname{diag}(\widehat{\omega}_1^2, \ldots, \widehat{\omega}_p^2)$, and together $(\widehat{B}, \widehat{\Omega})$ can be used to estimate the covariance matrix Σ (see Section 2.1).

For discrete data, we regress each node onto its parents set using multi-logit regression via the R package **nnet** (Venables and Ripley 2002). Note that for discrete data, instead of a coefficient matrix we get a 4-way array $\widehat{B} = (\widehat{\boldsymbol{\beta}}_{ij})$, where $\widehat{\boldsymbol{\beta}}_{ij}$ is a matrix and the (u, k) entry of this matrix is the influence that level k of X_i has on level u of X_j.

4. The sparsebn package

Based on the framework described in Section 3, **sparsebn** implements four structure learning algorithms that are each tailored to a specific type of data:

- Experimental, Gaussian data (Zhang 2016).
- Experimental, discrete data (Gu *et al.* 2018).
- Observational, Gaussian data (Aragam and Zhou 2015).
- Observational, discrete data (Gu *et al.* 2018).

By combining these approaches, **sparsebn** automatically handles datasets with mixed observational and experimental data. Each algorithm is implemented in C++ using **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013). In addition to the main algorithms, the package implements high-level methods for fitting, plotting, and manipulating graphical models.

Furthermore, **sparsebn** is actually a family of R packages, designed to be cross-compatible with minimal external dependencies. To date, **sparsebn** imports the following packages:

- **ccdrAlgorithm**, based on Aragam and Zhou (2015) and Fu and Zhou (2013).
- **discretecdAlgorithm** (Gu 2017), based on Gu *et al.* (2018).
- **sparsebnUtils** (Aragam, Gu, and Zhou 2017), for housing various common utilities and classes.

The idea is that the codebase for each algorithm is housed inside its own package, allowing for rapid development and convenient extensibility. This allows us to add new algorithms and features rapidly without significant dependency or compatibility constraints.

4.1. Speed and scalability improvements

sparsebn is designed to handle large, high-dimensional datasets with p potentially in the thousands. To illustrate this, Figure 2 provides a comparison of our methods with the PC algorithm from the **pcalg** package and the MMHC algorithm from the **bnlearn** package. Due to the higher computational workload for discrete algorithms in general, the results here are for Gaussian data only. Furthermore, although these numbers are intended to be illustrative, the interested reader may find more extensive simulations corroborating these results for both types of data in Aragam and Zhou (2015); Gu *et al.* (2018).

In order to provide a direct comparison, the times reported in Figure 2 reflect the total time to learn graphs of the same complexity (number of edges), even though **sparsebn** is capable of computing further into the solution path if desired. Note also that both the PC and MMHC algorithms output only a single graph, whereas our method outputs a solution path with multiple graph estimates. The results illustrate how **sparsebn** provides a more favorable scaling when p is large: For the largest graph with 1090 nodes, **sparsebn** is 17x faster compared to the PC algorithm in **pcalg** and 52x faster compared to the MMHC algorithm in **bnlearn**. These computational improvements are made possible by the use of an efficient block coordinate descent algorithm that leverages warm starts and active set updates; for more details see Section 3.2.

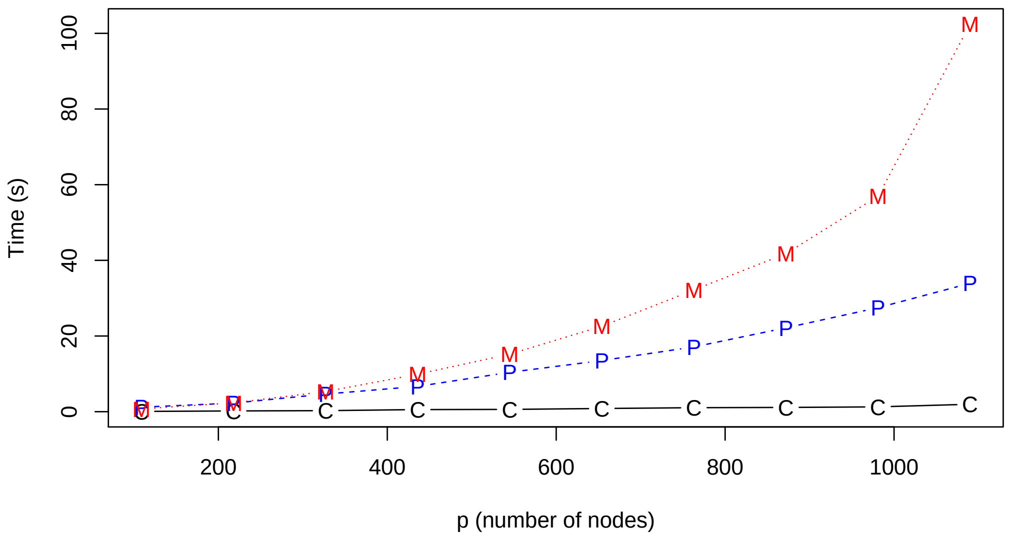


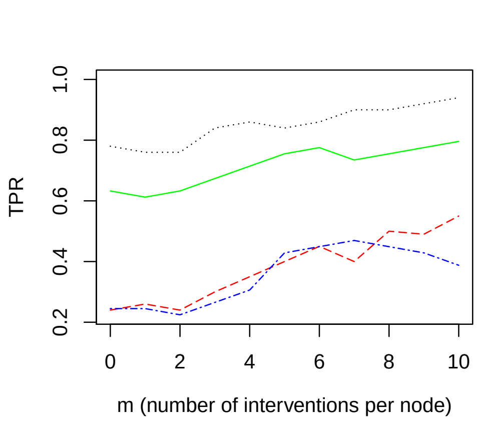
Figure 2: Timing comparison (in seconds). Each point represents the total runtime to execute an algorithm on simulated Gaussian data as a function of the number of nodes $p = 109k$ ($k = 1, \dots, 10$). The DAGs were constructed by tiling k independent copies of the pathfinder network ($p = 109$), and $n = 50$ samples were randomly generated for each dataset. (solid black line) C = CCDr algorithm implemented in **sparsebn**, (dashed blue line) P = PC algorithm implemented in **pcalg**, (dotted red line) M = MMHC algorithm implemented in **bnlearn**.

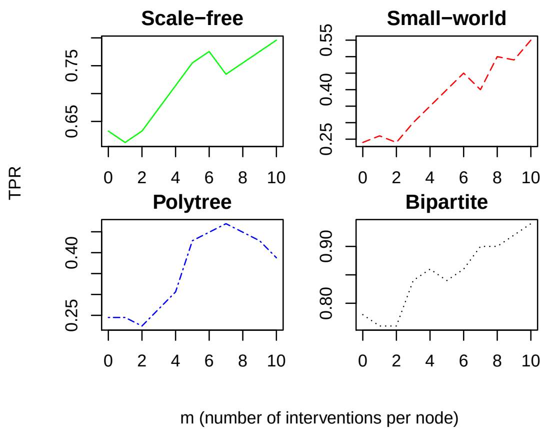
4.2. Experimental interventions

In addition to scalability, another feature that distinguishes **sparsebn** is its native support for mixed observational and experimental data. As discussed in Section 2.2, experimental interventions allow observationally equivalent DAGs to be distinguished, thereby uncovering the structure of the true causal DAG. To illustrate this, we ran two simulation experiments, reported in Figures 3 and 4. To keep things simple, we focus on discrete data. (Code to run these experiments on continuous data can be found at `https://github.com/itsrainingdata/sparsebn-reproduce`.) As in the previous subsection, a more thorough evaluation of the effect of interventions can be found in Gu *et al.* (2018); Zhang (2016); Fu and Zhou (2013).

Figure 3 illustrates how the accuracy of reconstruction improves as interventions are added to each node in the network. We can see that as we process more interventions per node (denoted by m) with n held fixed, the true positive rate (TPR) increases. Further analysis of these results indicates furthermore that the number of reversed edges decreases along with the number of false positives, and as a result the overall structural Hamming distance (SHD) – the total number of edge additions, deletions, and reversals needed to convert one directed graph into another – decreases.

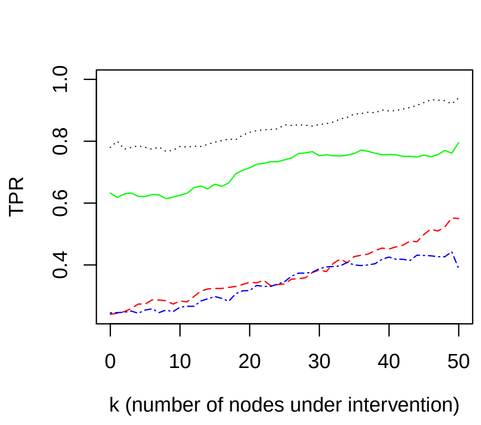
Figure 4 considers the more practical scenario in which only k ($k < p$) nodes in the network are under intervention, and over time more interventions on more nodes are able to be collected. This is common, for example, when reconstructing large networks in biological applications. Again, we see that overall the true positive rate increases as more nodes are under intervention, and further analysis shows that the SHD decreases as well.

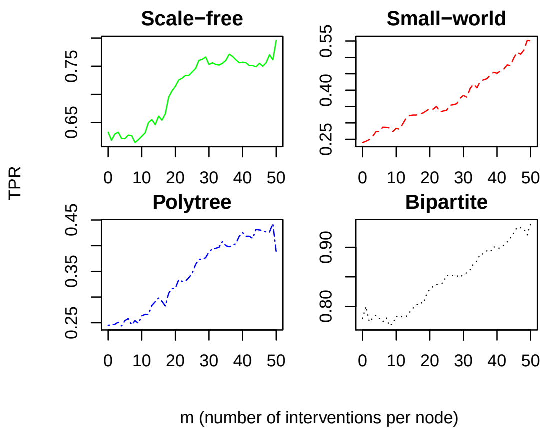




(A) (B)

Figure 3: The effect of interventions in learning discrete DAGs. For every node, m interventions are added to otherwise observational data with $n = 500$ and $p = 50$. (A) Scale-free network with 49 edges (solid green line), small-world network with 100 edges (dashed red line), polytree with 49 edges (dashed blue line), and bipartite graph with 50 edges (dashed black line); (B) Plot for each network individually.





(A) (B)

Figure 4: The effect of interventions in learning discrete DAGs. For each $k = 0, \ldots, 50$, $m = 10$ interventions for k randomly selected nodes are added to otherwise observational data with $n = 500$ and $p = 50$. Results are averaged over 20 random permutations of the order in which each node is intervened on as k is increased. (A) Scale-free network with 49 edges (solid green line), small-world network with 100 edges (dashed red line), polytree with 49 edges (dashed blue line), and bipartite graph with 50 edges (dashed black line); (B) Plot for each network individually.

Since n is held fixed in each simulation, the improvements observed here cannot be attributed to an increase in sample size, illustrating how **sparsebn** is able to improve estimation of the true causal graph under experimental interventions. In particular, reducing reversed edges that are observationally equivalent shows in an obvious way the utility of interventions for causal inference.

4.3. Functions

The main purpose of the **sparsebn** package is estimation of graphical models, which is accomplished through the methods prefaced with "`estimate.`". In this section, we present an overview of the main estimation methods. In addition, **sparsebn** includes methods for generating random graphs, simulating random data, visualization, and conversion between various graph classes, which are documented extensively in the package manual.

The main function is `estimate.dag`, which can be called as follows:

```
estimate.dag(data, lambdas = NULL, lambdas.length = 20, whitelist = NULL,
  blacklist = NULL, error.tol = 1e-04, max.iters = NULL,
  edge.threshold = NULL, concavity = 2, weight.scale = 1, convLb = 0.01,
  upperbound = 100, adaptive = FALSE, verbose = FALSE)
```

The main arguments are `data`, `lambdas`, and `lambdas.length`. By default, the `lambdas` argument is `NULL` and a standard sequence of $L = 20$ regularization parameters is generated. If desired, the user can pre-compute a vector of regularization parameters to be used instead, in which case this vector should be passed through the `lambdas` argument. If there is prior knowledge of (directed) edges that are known to be present in the network, these can be specified via the `whitelist` argument. Similarly, if there is prior knowledge of (directed) edges that are known to be absent from the network, these can be specified via the `blacklist` argument. The rest of the arguments control the convergence of the internal algorithms, and are intended for advanced users. This method returns a `sparsebnPath` object (Section 4.4), which stores the solution path described in Section 3.2.

It is important to bear in mind that the objects returned by `estimate.dag` are graphs, and in particular they do not include estimates of model parameters such as edge weights or conditional variances. To obtain these parameters, **sparsebn** includes the `estimate.parameters` method, which can be called as follows:

```
estimate.parameters(fit, data, ...)
```

where `fit` is the output of `estimate.dag` and `data` is the data to be used for parameter estimation.

In addition to estimating DAGs, **sparsebn** can estimate the precision and/or covariance matrix for multivariate Gaussian data. The nonzero entries in the precision matrix in particular yield the so-called *Gaussian graphical model*, which is an undirected graphical model for multivariate Gaussian data. This can be done via the `estimate.precision` and `estimate.covariance` methods:

```
estimate.covariance(data, ...)
estimate.precision(data, ...)
```

Internally, these methods call `estimate.dag` and use Equation 4 to compute the estimated precision (or covariance) matrix. The `...` argument here allows the user to specify any of the optional arguments from `estimate.dag`.

4.4. Data structures

sparsebn uses three different S3 classes in order to represent data (`sparsebnData`), graphs (`sparsebnFit`), and solution paths (`sparsebnPath`). For each of these classes, the usual generics are defined such as `print`, `summary`, and `plot`.

The `sparsebnData` class is used to represent both continuous and discrete data with experimental interventions. Observational data corresponds to the degenerate case where $\mathbf{X}$ does not contain any interventions, and is treated as such by the **sparsebn** package. The slots are:

- `data`: This is the original data as a data frame with n observations and p variables.
- `type`: Either `"continuous"` or `"discrete"`.
- `levels`: A list of levels for each variable. This is a list of length p whose jth component is a vector containing the levels of the jth variable.
- `ivn`: The list of interventions for each observations. This is a list of length n whose ith component is a vector of node names (or indices) that are under intervention for the ith observation.

The `sparsebnPath` class represents a solution path, which is the output of the main function `estimate.dag`. Internally, this is a `list` of `sparsebnFit` objects whose jth component corresponds to the jth value of λ in the solution path, $\lambda_{\max} > \lambda_1 > \cdots > \lambda_{\min}$. Since this class is essentially a wrapper for this list, it has no named slots.

The `sparsebnFit` class represents an individual graph estimate from a DAG learning algorithm. The graph itself is stored as an `edgeList` object in the `edges` slot, which is an internal implementation of a child-parent edge list. Alternatively, this graph can also be stored in a variety of other formats including `graphNEL` (from the **graph** package), `igraph` (from the **igraph** package), and `network` (from the **network** package) by using the `setGraphPackage` method (Section 4.5). The slots are:

- `edges`: A directed graph corresponding to the estimated network, stored internally as an `edgeList` by default.
- `nodes`: A vector of node names for the graph.
- `lambda`: The value of λ used to estimate the network.
- `nedge`: The total number of edges in the graph.
- `pp, nn, time`: The number of nodes, number of samples, and clock time to estimate this network (these are mainly used internally by the package).

4.5. Compatibility

Unfortunately, there is no consistent standard in R for storing and representing graphs. As a result, different domains have adopted different R packages as a *de facto* standard for graph and network representation. For example, in biology the **graph** package (Gentleman, Whalen, Huber, and Falcon 2011) seems to be the most popular, whereas in social science and demography the **network** package (Butts 2008) is more popular. In other domains, the **igraph** package (Csardi and Nepusz 2006) is popular, which has libraries in R, Python, and C.

For this reason, **sparsebn** does not provide its own mechanism for manipulating graphs, and instead provides cross-compatibility with each of these three packages. By default, all methods output graphs stored as an `edgeList` object, which is an internal class with little built-in functionality outside of being a storage mechanism for graph data. In order to make use of the extensive capabilities of the different graph packages in R, we have included the `setGraphPackage` method to allow users to set a global preference for which graph package to use. Once this preference is set, the full feature set of the selected package (e.g., plotting, manipulation, network statistics, etc.) becomes available to the user. We emphasize that the purpose of **sparsebn** is not to provide a new library for graph representation and visualization, but instead to provide algorithms for learning their structure. The manipulation of graphs is appropriately left to libraries designed explicitly for that purpose.

Furthermore, to allow cross-compatibility with existing packages for structure learning, we have provided methods to convert the output of **sparsebn** methods to **bnlearn**-compatible objects. Compatibility with the **pcalg** package is possible via the aforementioned **graph** package, which is the default representation used by **pcalg**.

The `setGraphPackage` method sets a *global* preference for the underlying class used to store graphs by the package. That is, all of the existing graph objects and any subsequent output will be coerced to the desired class. Generally speaking, this corresponds to the output of `estimate.dag` and any corresponding `sparsebnPath` objects. Alternatively, users can *manually* do this conversion on an object-by-object basis using the following methods:

- `to_igraph`: Conversion to and from `igraph` graphs from the **igraph** package;
- `to_graphNEL`: Conversion to and from `graphNEL` graphs from the **graph** package;
- `to_network`: Conversion to and from `network` graphs from the **network** package;
- `to_bn`: Conversion to and from `bn` graphs from the **bnlearn** package.

Each of these methods works on `sparsebnPath`, `sparsebnFit`, `edgeList` objects, in addition to any of the objects listed above.

Finally, the **sparsebn** package is compatible with the popular **Cytoscape** application (Shannon *et al.* 2003). This is a standalone graphical interface for visualizing and analyzing complex networks. This is accomplished via the `openCytoscape` method, which leverages the **RCy3** package (Shannon, Grimes, Kutlu, Bot, and Galas 2013) under the hood. In order to use this method, both **RCy3** and **Cytoscape** must be installed. For an example of this method in use, see Section 5.7.

4.6. Installation

sparsebn is an open-source package and is made freely available through CRAN. To install the latest stable version in R,

```
R> install.packages("sparsebn")
```

For advanced users, the development versions can be downloaded directly from GitHub. Using **devtools** (Wickham, Hester, Chang, RStudio, and R Core Team 2018), the entire suite of packages can be installed via

```
R> devtools::install_github(c("itsrainingdata/sparsebnUtils@dev",
+     "itsrainingdata/ccdrAlgorithm@dev", "gujyjean/discretecdAlgorithm@dev",
+     "itsrainingdata/sparsebn@dev"))
```

Note that before being released to CRAN, development versions may be unstable.

5. Example: Cytometry data

To illustrate the use of this package, we will use the flow cytometry dataset from Sachs, Perez, Pe'er, Lauffenburger, and Nolan (2005) as a working example in this section. The original dataset consists of $n = 7466$ observations of $p = 11$ continuous variables corresponding to different proteins and phospholipids in human immune system cells, and each observation indicates the measured level of each biomolecule in a single cell under different experimental interventions. A network consisting of all well-established causal interactions between these molecules has been constructed based on biological experiments and literature. This network is frequently used as a benchmark to assess the accuracy of BN learning algorithms on real data. Therefore, we refer to this as the consensus network in the sequel.

The consensus network is visualized in Figure 5, in which a directed edge indicates that a change in the level of the parent will cause a change in the level of the child. This is a relatively small network which we use in order to keep the exposition simple. More examples, including a discrete version of this dataset and applications to large networks with hundreds or thousands of nodes, will be discussed in Section 6.

First, we load this data:

```
R> library("sparsebn")
R> data("cytometryContinuous")
R> names("cytometryContinuous")
```

```
  [1] "dag"  "data" "ivn"
```

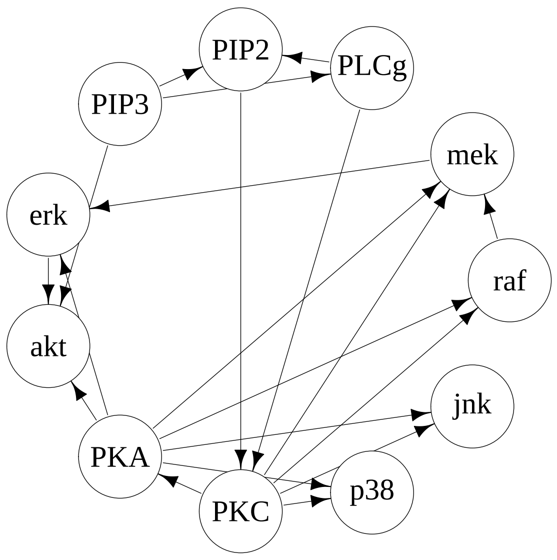


Figure 5: The consensus cytometry network (11 nodes, 17 edges). A topological ordering of this network is PIP3 $\prec$ PLCg $\prec$ PIP2 $\prec$ PKC $\prec$ PKA $\prec$ raf $\prec$ mek $\prec$ erk $\prec$ akt $\prec$ p38 $\prec$ jnk.

Note that this is not a `data.frame`, but instead a list of R objects that will be useful for this specific example. Each component of this list stores an important part of the experiment:

- `dag` is the consensus network with 11 nodes and 17 edges, as described above.
- `data` is raw data collected from these experiments.
- `ivn` is a list of interventions for each observation in the dataset.

To illustrate the use of this package, the rest of this section describes the main steps to learning Bayesian networks from data: (1) Loading the data, (2) Learning the structure, (3) Incorporating prior knowledge, (4) Exploring the solution path and estimated networks, (5) Estimating the parameters, (6) Selecting the regularization parameter, and (7) Visualizing and assessing the output.

5.1. Loading data

In order to distinguish different types of data – namely, experimental versus observational and continuous versus discrete – we use the `sparsebnData` class which wraps a `data.frame` into an object containing this auxiliary information. All of the methods implemented in **sparsebn** expect input as `sparsebnData`.

To use this class, we need two important pieces of information: The raw data as a `data.frame`, and a list of interventions for each observation in the dataset. If the dataset does not contain any interventions, then the latter can be omitted. In order to create a `sparsebnData` object from the cytometry data, we first extract the necessary objects from `cytometryContinuous`:

```
R> cyto.raw <- cytometryContinuous$data
R> cyto.ivn <- cytometryContinuous$ivn
```

Now we can create the required `sparsebnData` object:

```
R> cyto.data <- sparsebnData(cyto.raw, type = "continuous",
+    ivn = cyto.ivn)
```

Notice that we need to explicitly specify that the data is continuous (for discrete data, one would specify `type = "discrete"`).

Finally, for discrete data, we may wish to manually specify the levels of each variable, which can be done using the `levels` argument. When this argument is omitted, we attempt to automatically infer the levels. Note that in doing so, however, levels which are missing from the data will not be recognized by the `sparsebnData` constructor.

```
R> print(cyto.data)
```

```
    raf     mek  plc pip2  pip3  erk  akt  pka  pkc   p38   jnk
1: 3.27 2.58022 2.18 2.91 4.074 1.89 2.83 6.03 2.83 3.804 3.689
2: 3.58 2.80336 2.51 2.82 2.096 2.92 3.48 5.86 1.21 2.803 4.119
3: 4.08 3.78646 2.68 2.32 2.565 2.70 3.48 6.00 2.43 3.463 2.970
4: 4.29 4.41643 3.14 2.60 0.255 1.76 2.47 6.27 2.62 3.353 3.140
```

```
   5: 3.52 2.98568 1.65 2.28 3.211 3.05 3.83 5.72 1.54 3.246 4.398
  ---
7462: 3.89 2.51770 3.49 3.33 3.122 2.46 3.64 7.04 0.00 0.936 0.000
7463: 3.15 1.52823 2.88 3.10 2.701 3.89 4.21 6.83 0.00 2.284 0.000
7464: 3.34 1.50185 2.93 3.01 2.322 1.12 3.09 6.59 0.00 0.560 0.693
7465: 3.54 1.96009 1.75 3.03 2.715 3.47 3.72 6.70 3.80 7.231 0.892
7466: 3.42 0.00995 1.99 5.15 3.131 1.89 2.62 6.79 0.00 0.000 0.501

7466 total rows (7456 rows omitted)
Continuous data w/ interventions on 4863/7466 rows.
```

```
R> summary(cyto.data)
```

```
      raf            mek            plc            pip2
 Min.   :0.00   Min.   :0.00   Min.   :0.00   Min.   :0.00
 1st Qu.:3.43   1st Qu.:2.80   1st Qu.:2.24   1st Qu.:2.91
 Median :3.99   Median :3.28   Median :2.80   Median :3.97
 Mean   :4.09   Mean   :3.53   Mean   :2.88   Mean   :3.90
 3rd Qu.:4.63   3rd Qu.:4.17   3rd Qu.:3.30   3rd Qu.:5.15
 Max.   :8.44   Max.   :8.87   Max.   :8.73   Max.   :9.11
      pip3           erk            akt            pka
 Min.   :0.00   Min.   :0.00   Min.   :0.00   Min.   :0.00
 1st Qu.:2.26   1st Qu.:2.14   1st Qu.:3.15   1st Qu.:5.62
 Median :2.88   Median :2.84   Median :3.62   Median :6.11
 Mean   :2.82   Mean   :2.75   Mean   :3.79   Mean   :5.83
 3rd Qu.:3.49   3rd Qu.:3.47   3rd Qu.:4.28   3rd Qu.:6.62
 Max.   :7.15   Max.   :7.85   Max.   :8.18   Max.   :9.09
      pkc            p38            jnk
 Min.   :0.00   Min.   :0.00   Min.   :0.00
 1st Qu.:1.50   1st Qu.:2.96   1st Qu.:2.08
 Median :2.54   Median :3.42   Median :2.91
 Mean   :2.37   Mean   :3.53   Mean   :3.00
 3rd Qu.:3.16   3rd Qu.:3.90   3rd Qu.:3.97
 Max.   :7.38   Max.   :8.92   Max.   :8.46

7466 total rows (7456 rows omitted)
Continuous data w/ interventions on 4863/7466 rows.
```

Note that some of the observations were not under intervention – such datasets with mixed observational and experimental samples are automatically handled by the methods in this package.

5.2. Structure learning

To learn the structure of a Bayesian network from this data we use the `estimate.dag()` method, which runs the algorithm outlined in Section 3.2. To call this method using the default parameter settings, use:

```
R> cyto.learn <- estimate.dag(cyto.data)
R> print(cyto.learn)

sparsebn Solution Path
 11 nodes
 7466 observations
 20 estimates for lambda in [0.8641, 86.406]
 Number of edges per solution: 0-1-6-8-13-15-15-19-22-21-26-33-35-36-38-38-
 41-43-46-50

R> summary(cyto.learn)

sparsebn Solution Path
 11 nodes
 7466 observations
 20 estimates for lambda in [0.8641, 86.406]
 Number of edges per solution: 0-1-6-8-13-15-15-19-22-21-26-33-35-36-38-38-
 41-43-46-50

   lambda nedge
1  86.406     0
2  67.808     1
3  53.213     6
4  41.759     8
5  32.771    13
6  25.717    15
7  20.182    15
8  15.838    19
9  12.429    22
10  9.754    21
11  7.654    26
12  6.007    33
13  4.714    35
14  3.699    36
15  2.903    38
16  2.278    38
17  1.788    41
18  1.403    43
19  1.101    46
20  0.864    50
```

In addition to `data`, there are several optional parameters that can be passed to `estimate.dag`. The main arguments of interest are `lambdas` and `lambdas.length`, which allow the user to adjust the grid of regularization parameters $\lambda_{\max} > \lambda_1 > \cdots > \lambda_{\min}$ used by the algorithms (Section 3.2).

By default, `estimate.dag` produces a solution path of 20 estimates with the grid chosen adaptively to the data. This grid can be shortened or lengthened by specifying `lambdas.length`:

```
R> estimate.dag(cyto.data, lambdas.length = 50)

sparsebn Solution Path
 11 nodes
 7466 observations
 50 estimates for lambda in [0.8641, 86.406]
 Number of edges per solution: 0-0-1-1-2-6-6-7-8-11-12-12-13-15-15-16-16-17-
 19-19-20-21-21-21-21-24-27-28-31-35-35-35-34-36-36-37-38-38-38-38-39-41-42-
 42-44-45-45-47-48-50
```

For even more fine-tuning, the `lambdas` argument allows the user to explicitly input their own grid. For convenience we have included the `generate.lambdas` method, which provides a mechanism for generating grids of arbitrary lengths on either a linear or log scale. To generate a grid with a linear scale, use `scale = "linear"`:

```
R> cyto.lambdas <- generate.lambdas(lambda.max = 10, lambdas.ratio = 0.001,
+    lambdas.length = 10, scale = "linear")
R> cyto.lambdas

[1] 10.00  8.89  7.78  6.67  5.56  4.45  3.34  2.23  1.12  0.01
```

To use a log scale, use `scale = "log"`:

```
R> cyto.lambdas <- generate.lambdas(lambda.max = 10, lambdas.ratio = 0.001,
+    lambdas.length = 10, scale = "log")
R> cyto.lambdas

[1] 10.0000  4.6416  2.1544  1.0000  0.4642  0.2154  0.1000  0.0464
[9]  0.0215  0.0100
```

This grid can also be generated manually, although this is not recommended. To run the algorithm using `cyto.lambdas` (output suppressed below):

```
R> estimate.dag(cyto.data, lambdas = cyto.lambdas)
```

Another argument of interest is `edge.threshold`, which is another way to specify when the algorithm terminates. Specifically, if any point on the solution path contains an estimate with more than `edge.threshold` edges, the algorithm will terminate immediately and return what has been estimated up to that point. This makes our methods *anytime algorithms*, in the sense that they can be interrupted at anytime while still producing valid output. This is convenient when running tests on very large graphs.

5.3. Prior knowledge

In some contexts, users may have prior knowledge regarding edges that must be present or absent from the network. For example, it may already be known that PIP3 regulates PIP2 (see Figure 5). In this case, estimation of the underlying network can be substantially improved by incorporating this information into the estimation procedure. With the **sparsebn** package, this can be done via *whitelists* and *blacklists*, which specify edges that must be present and absent, respectively.

For example, to specify a known relationship between PIP3 and PIP2, we can create a whitelist as follows:

```
R> whitelist <- matrix(c("pip3", "pip2"), nrow = 1)
R> estimate.dag(cyto.data, whitelist = whitelist)
```

The `whitelist` argument should be a two-column matrix, where the first column stores parents and the second stores children (i.e., a from-to adjacency list):

```
R> whitelist
```

```
     [,1]   [,2]
[1,] "pip3" "pip2"
```

Thus, this whitelist ensures that the edge PIP3→PIP2 will be present in the final estimates.

Similarly, we can specify a blacklist, which stores edges that are known to be absent. For example, we can forbid any edges between RAF and MEK as follows:

```
R> blacklist <- rbind(c("raf", "jnk"), c("jnk", "raf"))
R> estimate.dag(cyto.data, blacklist = blacklist)
```

As with the whitelist, the blacklist should be a two-column matrix. Note that we specify *both* directions RAF→MEK and MEK→RAF. If it is known that the direction can only go in one direction, then a single direction may be specified instead.

Blacklists are useful for specifying known root and leaf nodes in a Bayesian network. In the cytometry network, PIP3 is a root node (i.e., it has no parents). Thus, we can forbid any edges pointing *into* PIP3. Similarly, JNK, P38, and AKT are leaf nodes (i.e., they have no children), so we can forbid any edges pointing *away* from all three nodes. To specify this, we make use of the `specify.prior` function, which automatically builds an appropriate blacklist given the names of the root and leaf nodes. Any number of root and/or leaf nodes can be specified.

```
R> blacklist <- specify.prior(roots = "pip3", leaves = c("jnk", "p38", "akt"),
+    nodes = names(cyto.data$data))
R> estimate.dag(cyto.data, blacklist = blacklist)
```

Finally, whitelists and blacklists can be combined arbitrarily, as long as they are consistent in the sense that no edge appearing in the whitelist appears in the blacklist, and vice versa. This allows for a powerful specification of prior knowledge in learning networks from data.

5.4. Solution paths

The output of `estimate.dag` is a `sparsebnPath` object, which stores the entire solution path that is returned by the method. This is similar in spirit to the **glasso** package (Friedman, Hastie, and Tibshirani 2018), however, instead of storing each estimate as an R matrix we use the internal class `sparsebnFit`. Since `sparsebnPath` objects also inherit from the `list` class in base R, we can inspect the first solution using ordinary R indexing. Note that for `sparsebnFit` objects, the `print` and `summary` methods are identical, so the output below is shown only once.

```
R> print(cyto.learn[[1]])
R> summary(cyto.learn[[1]])

CCDr estimate
7466 observations
lambda = 86.4060183089118

DAG:
<Empty graph on 11 nodes.>
```

The first estimate will always be the empty graph, which is a consequence of how we employ warm starts in the solution path. The third estimate, for example, shows a bit more structure:

```
R> print(cyto.learn[[3]])
R> summary(cyto.learn[[3]])

CCDr estimate
7466 observations
lambda = 53.2129918008817

DAG:
[raf]   mek
[mek]
[plc]
[pip2]  plc
[pip3]
[erk]   akt
[akt]
[pka]   p38
[pkc]
[p38]   pkc
[jnk]   pkc
```

Each row in the output above corresponds to a child node – indicated by the square brackets – with its parents listed to the right without brackets. Formally, this is an adjacency list ordered by children. For large graphs, explicit output of the parental structure as shown here is omitted by default, however, this behavior can be overridden via the `maxsize` argument. Alternatively, we can retrieve the adjacency matrix, $\{I(\widehat{\beta}_{ij} \neq 0)\}_{p\times p}$, for this estimate:

```
R> get.adjacency.matrix(cyto.learn[[3]])
```

```
11 x 11 sparse Matrix of class "dgCMatrix"

raf  . . . . . . . . . . .
mek  1 . . . . . . . . . .
plc  . . . 1 . . . . . . .
pip2 . . . . . . . . . . .
pip3 . . . . . . . . . . .
erk  . . . . . . . . . . .
akt  . . . . . 1 . . . . .
pka  . . . . . . . . . . .
pkc  . . . . . . . . . 1 1
p38  . . . . . . . 1 . . .
jnk  . . . . . . . . . . .
```

Note the use of the **Matrix** package (Bates and Mächler 2019), which reduces the memory footprint on large graphs.

Finally, for large graphs, it may be desirable to inspect a subset of nodes, which can be done using the `show.parents` method:

```
R> show.parents(cyto.learn[[3]], c("raf", "pip2"))

[raf]   mek
[pip2]  plc
```

5.5. Parameter estimation

It is important to note that the output of `estimate.dag` is a sequence of *graphs*, i.e., no parameters (edge weights, variances, etc.) have been estimated at this stage. The next step is to estimate the values of the parameters associated with the underlying joint distribution. This is easy to do:

```
R> cyto.param <- estimate.parameters(cyto.learn, data = cyto.data)
```

The output is a list with each entry containing a component for the weighted adjacency matrix (`coefs`) and a diagonal matrix for the conditional variances (`vars`). For example, the coeffients of the third estimate in the solution path with six edges are (rounded to two decimal places):

```
R> cyto.param[[3]]$coefs

11 x 11 sparse Matrix of class "dgCMatrix"

 [1,] .    . . .    . .     . .    . .   .
 [2,] 1.05 . . .    . .     . .    . .   .
 [3,] .    . . 1.26 . .     . .    . .   .
 [4,] .    . . .    . .     . .    . .   .
 [5,] .    . . .    . .     . .    . .   .
 [6,] .    . . .    . .     . .    . .   .
 [7,] .    . . .    . 0.725 . .    . .   .
 [8,] .    . . .    . .     . .    . .   .
 [9,] .    . . .    . .     . .    . 1.3 1.13
[10,] .    . . .    . .     . 1.36 . .   .
[11,] .    . . .    . .     . .    . .   .
```

Similarly, we can inspect the estimated conditional variances:

```
R> Matrix::diag(cyto.param[[3]]$vars)
```

```
[1] 1.165 2.632 1.583 2.111 0.992 0.684 0.968 8.425 1.831 1.506 1.676
```

Although the `vars` argument is a diagonal matrix, we have invoked the `diag` method above purely to save space.

For Gaussian data, we can also use Equation 3 to estimate the implied covariance matrix for each solution in the solution path. This is implemented via the `get.covariance` method (see also `get.precision` for computing the precision, or inverse covariance, matrix). If the user is only interested in the covariance matrix (resp. precision matrix) and not the underlying DAG, then this extra step can be skipped by directly using the `estimate.covariance` method (resp. `estimate.precision`).

5.6. Model selection

Unlike existing methods, the output of `estimate.dag` is a solution path with multiple estimates of increasing complexity, indexed by the regularization parameter. Thus, it is important to be able to pick out estimates for inspection and further exploration. For *ad hoc* exploration, the `select` method is useful: This allows one to select an estimate from a `sparsebnPath` object based on the number of edges, the regularization parameter λ, or the index j. When selecting by the number of edges or by λ, fuzzy matching is used by default so that the closest match is returned to within a given tolerance. Selecting by index is equivalent to subsetting as usual with the subset operator `‘[[‘`. To save space, the output of the following code is suppressed:

```
R> select(cyto.learn, edges = 8)
R> select(cyto.learn, edges = 10)
```

In the first line above, an exact match is returned. In the second line, the closest match is returned since there is no graph with exactly 10 edges in the solution path.

```
R> select(cyto.learn, lambda = 41.75)
R> select(cyto.learn, lambda = 41.7)
```

In both of the above examples, the closest match is returned.

```
R> select(cyto.learn, index = 4)
R> cyto.learn[[4]]
```

In the both lines above, an exact match is returned. Note that the second line is equivalent to the first.

For practical applications, one is often concerned with selecting an optimal value of λ. In high-dimensions, optimal selection of λ for finite samples is an open problem, and past work has shown that both the prediction oracle and cross-validated choices perform poorly (Meinshausen and Bühlmann 2006; Fu and Zhou 2013). Instead, Fu and Zhou (2013) suggest a practical method for selecting λ based on a trade-off between the increase in log-likelihood and the increase in complexity between solutions. This method is implemented in **sparsebn** via the method `select.parameter`:

```
R> selected.lambda <- select.parameter(cyto.learn, cyto.data)
R> selected.lambda

[1] 8
```

`select.parameter` returns the optimal index according to this method, in this case $j = 8$, corresponding to a value of $\lambda \approx 15.83806$ and an estimated network of 19 edges.

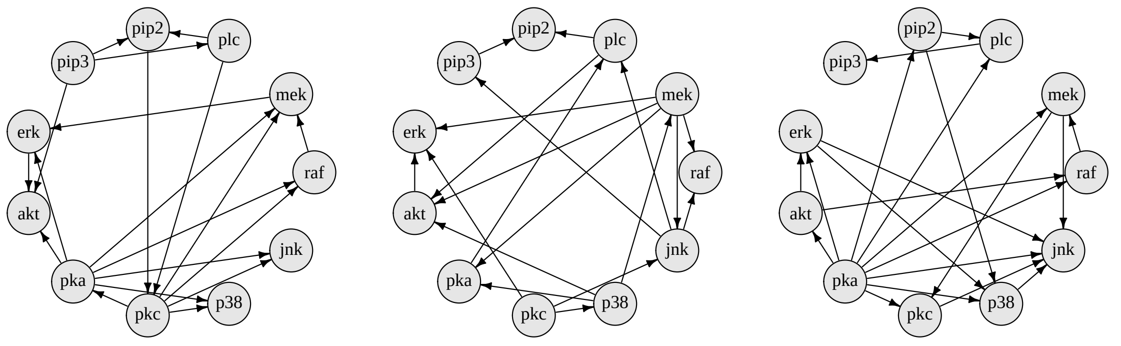
5.7. Visualization

In order to visualize graphs estimated by the **sparsebn** package, we make use of the visualization capabilities of existing graph packages (see Section 4.5). By default, **sparsebn** uses the popular **igraph** package:

```
R> getPlotPackage()

[1] "igraph"
```

In addition to **igraph**, **sparsebn** is also compatible with the plotting features of the **graph** (via **Rgraphviz**, Hansen *et al.* 2008) and **network** packages. It is easy to change which package is used for plotting:

```
R> setPlotPackage("network")
R> getPlotPackage()

[1] "network"

R> setPlotPackage("graph")
R> getPlotPackage()

[1] "graph"

R> setPlotPackage("igraph")
R> getPlotPackage()

[1] "igraph"
```



(a) Consensus network. (b) Continuous network. (c) Discrete network.

Figure 6: Cleaned up plots of (a) The consensus cytometry network, (b) The learned network based on continuous data, and (c) The learned network based on discretized data. Both learned networks were estimated using `estimate.dag`.

This allows the user complete flexibility over which R package is used for storing data and for visualizing data. In fact, **sparsebn** even allows one package to be used for visualization and a different package for storage. For large graphs, it is helpful to use a different set of defaults, provided in a separate method, `plotDAG`, which can be used for quick plotting (see, for example, Figure 1).

Visualizing the full solution path is easy: Simply call `plot(cyto.learn)`. In order to ensure flexibility, we maintain all of the defaults used by the selected package for the `plot` method. It is easy, however, to customize the appearance of the plots if desired. For example, if we would like to compare the consensus cytometry network and the estimated network side by side, we can adjust the arguments to `plot` as follows:

```
R> plot(cytometryContinuous$dag,
+    layout = igraph::layout_(to_igraph(cytometryContinuous$dag),
+      igraph::in_circle()),
+    vertex.label = names(cytometryContinuous$dag), vertex.size = 30,
+    vertex.label.color = gray(0), vertex.color = gray(0.9),
+    edge.color = gray(0), edge.arrow.size = 0.5)
R> plot(cyto.learn[[selected.lambda]],
+    layout = igraph::layout_(to_igraph(cytometryContinuous$dag),
+      igraph::in_circle()),
+    vertex.label = get.nodes(cyto.learn), vertex.size = 30,
+    vertex.label.color = gray(0), vertex.color = gray(0.9),
+    edge.color = gray(0), edge.arrow.size = 0.5)
```

The output can be seen in Figures 6a and 6b.

Finally, any object produced by **sparsebn** can be sent to the external application **Cytoscape** (`http://www.cytoscape.org/`) via the `openCytoscape` method. To use this method, **Cytoscape** must already be open and running in the background, and the **RCy3** package (Ono, Muetze, Kolishovski, Shannon, and Demchak 2015) from **Bioconductor** must also be installed.

6. Further examples

In this section we provide three more examples of the functionality of **sparsebn**: (1) An example with discrete data, (2) A benchmark network from the Bayesian network repository, and (3) A gene expression dataset with $p = 5000$ nodes.

6.1. Discrete cytometry data

In the previous section we used the cytometry network as an instructional example based on the original, continuous dataset. Sachs *et al.* (2005) also provided a cleaned, discretized version of this dataset which can be used to illustrate how these methods apply to discrete data. After cleaning and pre-processing, the original continuous measurements were binned into one of three levels: *low* = 0, *medium* = 1, or *high* = 2. Due to the pre-processing, the discrete data contains fewer observations ($n = 5400$) compared to the raw, continuous data.

To use this data, we start by loading it as usual:

```
R> library("sparsebn")
R> data("cytometryDiscrete")
```

The code to estimate this graph is essentially the same as in Section 5. For completeness, we present only the essential steps here. As before, the first step is to pass the data into a `sparsebnData` object:

```
R> cyto.data <- sparsebnData(cytometryDiscrete$data, type = "discrete",
+    ivn = cytometryDiscrete$ivn)
R> cyto.data
```

```
      raf mek plc pip2 pip3 erk akt pka pkc p38 jnk
   1:   0   0   0    1    2   1   0   2   0   1   0
   2:   0   0   0    0    2   2   1   2   0   1   0
   3:   0   0   1    1    2   1   0   2   1   0   0
   4:   0   0   0    0    2   1   0   2   0   2   0
   5:   0   0   0    0    2   1   0   2   0   0   0
  ---
5396:   0   0   0    0    1   1   0   1   1   0   0
5397:   0   0   0    0    0   1   1   0   0   1   1
5398:   0   0   0    0    1   1   0   1   1   0   0
5399:   0   1   0    0    0   0   0   1   1   0   0
5400:   1   1   0    0    1   1   0   1   1   1   0

5400 total rows (5390 rows omitted)
Discrete data w/ interventions on 3600/5400 rows.
```

One of the main purposes of the `sparsebnData` class is to encode all of the necessary information needed to run the main algorithms; now that the data has been converted into a `sparsebnData` object, the user will notice almost no difference between the code in Section 5 and the sequel.

To estimate a DAG based on the discrete data, use `estimate.dag`:

```
R> cyto.learn <- estimate.dag(cyto.data)
R> cyto.learn

sparsebn Solution Path
 11 nodes
 5400 observations
 8 estimates for lambda in [1.791, 9.7712]
 Number of edges per solution: 0-6-9-13-15-21-30-38
```

It is easy to adjust lambdas and lambdas.length as before. Note also the difference between the number of solutions here and in Section 5.2; this is a consequence of the stopping criterion used for discrete data (see also Section 3.2). For comparison with the network selected in Section 5.6 which had 19 edges, we use select to choose the estimate that is closest in complexity; the closest such network in the present case has 21 edges and is shown in Figure 6c. The code to reproduce this figure is below:

```
R> plot(select(cyto.learn, edges = 19),
+    layout = igraph::layout_(to_igraph(cytometryContinuous$dag),
+      igraph::in_circle()),
+    vertex.label = get.nodes(cyto.learn), vertex.size = 30,
+    vertex.label.color = gray(0), vertex.color = gray(0.9),
+    edge.color = gray(0), edge.arrow.size = 0.5)
```

To estimate the parameters associated with the multi-logit model, use estimate.parameters:

```
R> cyto.param <- estimate.parameters(cyto.learn, data = cyto.data)
```

As the parameter space for the multi-logit model is much larger than for the Gaussian model, the output is much more complicated. For example, the node RAF has a single parent PKA in the DAG selected above, and so the parameter space for this node is a 2×3 matrix:

```
R> cyto.param[[5]][["raf"]]

     (Intercept)      pka_1     pka_2
    1   0.4219942 -0.9457959 -2.282747
    2   1.8258925 -3.6595227 -5.034717
```

For each extra parent, there will be two more columns added to this matrix, owing to the fact that each variable has three levels. There is one such matrix for each node (see Section 3.3 for details on the estimated parameters of the multi-logit model).

6.2. The pathfinder network

In order to illustrate this package on a larger network, we will reconstruct the pathfinder network (Heckerman *et al.* 1992). The pathfinder network has 109 nodes and 195 edges and is part of the Bayesian network repository at http://www.bnlearn.com/bnrepository/, a centralized repository of benchmark networks for testing structure learning algorithms.

We first load the dataset:

```
R> data("pathfinder")
R> dat <- sparsebnData(pathfinder$data, type = "c")
```

The data was generated from a Gaussian SEM with $\beta_{ij} = 1$ whenever $\beta_{ij} \neq 0$ and $\omega_j^2 = 1$ for each j. By Equation 3, we were able to compute the implied covariance matrix and use **mvtnorm** to generate samples from this distribution (Genz *et al.* 2019). For this example, $n = 1000$ samples were drawn.

For illustrative purposes, we will estimate a longer solution path with 50 DAGs:

```
R> nn <- num.samples(dat)
R> lambdas <- generate.lambdas(sqrt(nn), 0.05, lambdas.length = 50,
+    scale = "linear")
R> dags <- estimate.dag(data = dat, lambdas = lambdas, edge.threshold = 500,
+    verbose = FALSE)
R> dags

 109 nodes
 1000 observations
 50 estimates for lambda in [1.5811, 31.6228]
 Number of edges per solution: 0-17-22-29-34-37-39-50-58-59-63-63-64-64-
 65-78-103-108-108-108-108-108-108-108-108-108-108-108-108-108-113-115-
 119-121-121-121-124-130-135-137-139-144-153-166-180-189-206-219-249-370
```

The choice of $\lambda_{\max} = n^{1/2}$ in `generate.lambdas` is important as it guarantees that the first solution will be the empty graph (see Aragam and Zhou 2015, Section 5.3). We use a linear scale for the grid and we set `edge.threshold = 500` in order to terminate the algorithm early if any estimate has more than 500 edges (although note that in this case this constraint never becomes active).

Furthermore, there are no difficulties if the data is high-dimensional ($p > n$). Let us extract the first 50 rows of $\mathbf{X}$, so that $p = 109 > n = 50$:

```
R> dat <- sparsebnData(pathfinder$data[1:50, ], type = "c")
R> nn <- num.samples(dat)
R> lambdas <- generate.lambdas(sqrt(nn), 0.05, lambdas.length = 50,
+    scale = "linear")
R> dags <- estimate.dag(data = dat, lambdas = lambdas, edge.threshold = 500,
+    verbose = FALSE)
R> dags

sparsebn Solution Path
 109 nodes
 50 observations
 43 estimates for lambda in [1.3132, 7.0711]
 Number of edges per solution: 0-16-21-23-28-39-39-42-44-49-53-58-63-64-67-
 69-71-80-81-84-88-92-99-102-107-111-116-116-118-123-127-134-142-149-161-
 173-193-217-229-262-311-382-477
```

Note that in this case the algorithm terminates just shy of the full 50 estimates, since the 44th solution apparently exceeds the threshold of 500 edges. Finally, for this high-dimensional example, there is no noticeable slowdown compared to the previous case with $n = 1000$.

6.3. Large networks

To conclude the examples, we finish with an application to the estimation of very large networks using gene expression data collected from 129 late-onset Alzheimer's disease (LOAD) patients and 101 healthy patients (Zhang *et al.* 2013). The data consists of post-mortem whole-genome gene-expression profiles from autopsied brain tissues and in particular constitutes a purely observational sample. In total there are three datasets, each corresponding to tissue samples collected from a different brain region: (1) Dorsolateral prefrontal cortex (PFC), (2) Visual cortex (VC) and (3) Cerebellum (CR). Since ground truth networks have not been established for these datasets, the main purpose of this example is to illustrate how **sparsebn** scales to real world, high-dimensional data with thousands of variables.

Each dataset contains expression measurements from $39,280$ probes, of which we extracted the first $p = 5,000$ columns for use in this example. For this example, we focused on LOAD patients only, so $n = 129$. We then used `estimate.dag` to learn a solution path from each dataset, and kept track of the time. In our experiments, it took a little over four minutes for our package to estimate each solution path, assuming a maximum of 5000 edges. For example, assuming the output has been stored in a variable called `dags`, the resulting output for the PFC data is

```
R> dags

sparsebn Solution Path
 5000 nodes
 129 observations
 15 estimates for lambda in [2.9973, 11.3578]
 Number of edges per solution: 0-79-523-1297-2200-2999-3675-4315-4929-4946-
 4957-4963-4984-4988-5000
```

The output for the remaining two datasets is similar, consisting of 15 DAGs for the VC data and 16 DAGs for the CR data. Code to reproduce this experiment can be found in supplementary file `v91i11.R` (and at `https://github.com/itsrainingdata/sparsebn-reproduce`).

Some comments on the timing are in order. The `edge.threshold` argument was used here to terminate the algorithm at 5000 edges – in practice, of course, we do not know the true number of edges and so this is meant to be purely illustrative. Moreover, since this parameter can be set arbitrarily high, the algorithm can in principle run for arbitrarily long. In fact, one of the advantages of our methods is that they can be interrupted at anytime, bearing in mind that the result may be a *sub*graph of the true graph. The `edge.threshold` parameter allows the user to incorporate prior knowledge of the sparsity (or lack thereof) of the underlying graph.

7. Conclusion

sparsebn is a fast, modern package for learning the structure of sparse Bayesian networks. By leveraging recent trends in nonconvex optimization, sparse regularization, and causal inference, we are able to scale structure learning to problems containing high-dimensional data with thousands of variables and experimental interventions. This fills a gap within existing software packages for learning Bayesian networks, which already provide excellent coverage for traditional problems with large samples and without interventions. All of the code to reproduce the results presented here is available on GitHub, along with the source code of the **sparsebn** package, which is available on CRAN.

Given the nonconvex nature of the minimization problems here, one future direction is to incorporate stochastic optimization into our package to enhance its global search ability. For example, stochastic gradient descent may be used in the algorithm for discrete Bayesian networks, which will reduce the computational complexity as well. We are also interested in developing divide-and-conquer strategies for ultra-large graphs, say on the scale of 10^5 nodes. Along these lines, we are also exploring parallel and distributed implementations of these algorithms. Finally, our package can be combined with various post-learning functions for multi-stage learning of causal networks. Given the DAG learned from **sparsebn**, one may further infer causal relations in a subgraph of interest by making additional causal assumptions or with new experimental data.

Acknowledgements

This work was supported by NSF grants IIS-1546098 and DMS-1055286 to Q.Z. The authors thank Dacheng Zhang for helpful discussions and computational assistance.

References

Aragam B, Amini AA, Zhou Q (2015). "Learning Directed Acyclic Graphs with Penalized Neighbourhood Regression." *arXiv 1511.08963*, arXiv.org E-Print Archive. URL `https://arxiv.org/abs/1511.08963`.

Aragam B, Gu J, Zhou Q (2017). "Learning Large-Scale Bayesian Networks with the **sparsebn** Package." *arXiv 1703.04025*, arXiv.org E-Print Archive. URL `https://arxiv.org/abs/1703.04025`.

Aragam B, Zhou Q (2015). "Concave Penalized Estimation of Sparse Gaussian Bayesian Networks." *Journal of Machine Learning Research*, **16**, 2273–2328. `doi:10.1214/12-aos1017supp`.

Bates D, Mächler M (2019). ***Matrix**: Sparse and Dense Matrix Classes and Methods*. R package version 1.2-17, URL `https://CRAN.R-project.org/package=Matrix`.

Boettcher S, Dethlefsen C (2003). "**deal**: A Package for Learning Bayesian Networks." *Journal of Statistical Software*, **8**(1), 1–40. ISSN 1548-7660. `doi:10.18637/jss.v008.i20`.

Bouckaert RR (1993). "Probabilistic Network Construction Using the Minimum Description Length Principle." In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty: European Conference ECSQARU '93*, volume 747 of *Lecture Notes in Computer Science*, pp. 41–48. Springer-Verlag.

Bühlmann P, Kalisch M, Meier L (2014). "High-Dimensional Statistics with a View Toward Applications in Biology." *Annual Review of Statistics and Its Application*, **1**. doi:10.1146/annurev-statistics-022513-115545.

Buntine W (1991). "Theory Refinement on Bayesian Networks." In *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 52–60. Morgan Kaufmann.

Butts C (2008). "**network**: A Package for Managing Relational Data in R." *Journal of Statistical Software*, **24**(1), 1–36. ISSN 1548-7660. doi:10.18637/jss.v024.i02.

Chen YP, Wang ZX, Chen L, Liu X, Tang LL, Mao YP, Li WF, Lin AH, Sun Y, Ma J (2015). "A Bayesian Network Meta-Analysis Comparing Concurrent Chemoradiotherapy Followed by Adjuvant Chemotherapy, Concurrent Chemoradiotherapy Alone and Radiotherapy Alone in Patients with Locoregionally Advanced Nasopharyngeal Carcinoma." *Annals of Oncology*, **26**(1), 205–211.

Chickering DM, Heckerman D (1997). "Efficient Approximations for the Marginal Likelihood of Bayesian Networks with Hidden Variables." *Machine Learning*, **29**, 181–212. doi:10.1023/a:1007469629108.

Chickering DM, Heckerman D, Meek C (2004). "Large-Sample Learning of Bayesian Networks Is NP-Hard." *The Journal of Machine Learning Research*, **5**, 1287–1330. doi:10.1016/b978-1-55860-377-6.50079-7.

Colombo D, Maathuis MH, Kalisch M, Richardson TS (2012). "Learning High-Dimensional Directed Acyclic Graphs with Latent and Selection Variables." *The Annals of Statistics*, **40**(1), 294–321. doi:10.1214/11-aos940.

Cooper GF, Herskovits E (1992). "A Bayesian Method for the Induction of Probabilistic Networks from Data." *Machine Learning*, **9**, 309–347. doi:10.1007/bf00994110.

Cooper GF, Yoo C (1999). "Causal Discovery from a Mixture of Experimental and Observational Data." In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 116–125. Morgan Kaufmann Publishers Inc.

Csardi G, Nepusz T (2006). "The **igraph** Software Package for Complex Network Research." *InterJournal*, **Complex Systems**, 1695. doi:10.1142/s0219525914500064.

Cussens J, Haws D, Studený M (2017). "Polyhedral Aspects of Score Equivalence in Bayesian Network Structure Learning." *Mathematical Programming*, **164**(1-2), 285–324. doi:10.1007/s10107-016-1087-2.

Dejaeger K, Verbraken T, Baesens B (2013). "Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers." *IEEE Transactions on Software Engineering*, **39**(2), 237–257. doi:10.1109/tse.2012.20.

Dobson AJ, Barnett A (2008). *An Introduction to Generalized Linear Models*. CRC Press.

Eddelbuettel D (2013). *Seamless R and C++ Integration with* ***Rcpp***. Springer-Verlag, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Ellis B, Wong WH (2008). "Learning Causal Bayesian Network Structures from Experimental Data." *Journal of the American Statistical Association*, **103**, 778–789. doi:10.1198/016214508000000193.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. doi:10.1198/016214501753382273.

Friedman J, Hastie T, Höfling H, Tibshirani R (2007). "Pathwise Coordinate Optimization." *The Annals of Applied Statistics*, **1**(2), 302–332. doi:10.1214/07-aoas131.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of statistical software*, **33**(1), 1. doi:10.18637/jss.v033.i01.

Friedman J, Hastie T, Tibshirani R (2018). ***glasso**: Graphical Lasso – Estimation of Gaussian Graphical Models*. R package version 1.10, URL https://CRAN.R-project.org/package=glasso.

Fu F, Zhou Q (2013). "Learning Sparse Causal Gaussian Networks With Experimental Intervention: Regularization and Coordinate Descent." *Journal of the American Statistical Association*, **108**(501), 288–300. doi:10.1080/01621459.2012.754359.

Gámez JA, Mateo JL, Puerta JM (2011). "Learning Bayesian Networks by Hill Climbing: Efficient Methods Based on Progressive Restriction of the Neighborhood." *Data Mining and Knowledge Discovery*, **22**(1-2), 106–148. doi:10.1007/s10618-010-0178-6.

Gao B, Cui Y (2015). "Learning Directed Acyclic Graphical Structures with Genetical Genomics Data." *Bioinformatics*, p. btv513.

Garvey MD, Carnovale S, Yeniyurt S (2015). "An Analytical Framework for Supply Network Risk Propagation: A Bayesian Network Approach." *European Journal of Operational Research*, **243**(2), 618–627. doi:10.1016/j.ejor.2014.10.034.

Gentleman R, Whalen E, Huber W, Falcon S (2011). ***graph**: A Package to Handle Graph Data Structures*. R package version 1.30.0, URL https://CRAN.R-project.org/package=graph.

Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2019). ***mvtnorm**: Multivariate Normal and t Distributions*. R package version 1.0-11, URL https://CRAN.R-project.org/package=mvtnorm.

Gu J (2017). ***discretecdAlgorithm**: Coordinate-Descent Algorithm for Learning Sparse Discrete Bayesian Networks*. R package version 0.0.5, URL https://CRAN.R-project.org/package=discretecdAlgorithm.

Gu J, Fu F, Zhou Q (2018). "Penalized Estimation of Directed Acyclic Graphs From Discrete Data." *Statistics and Computing*. doi:10.1007/s11222-018-9801-y.

Hansen KD, Gentry J, Long L, Gentleman R, Falcon S, Hahne F, Sarkar D (2008). ***Rgraphviz**: Provides Plotting Capabilities for R Graph Objects.* R package version 1.18.1, URL https://CRAN.R-project.org/package=Rgraphviz.

Heckerman D, Geiger D, Chickering DM (1995). "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data." *Machine Learning*, **20**, 197–243. doi:10.1007/bf00994016.

Heckerman D, Horvitz E, Nathwani B (1992). "Toward Normative Expert Systems: Part I, the Pathfinder Project. Knowledge Systems Laboratory, Medical Computer Science."

Herskovits E, Cooper G (1990). "Kutató: An Entropy-Driven System for Construction of Probabilistic Expert Systems from Databases." In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 54–62.

Højsgaard S (2012). "Graphical Independence Networks with the **gRain** Package for R." *Journal of Statistical Software*, **46**(1), 1–26. ISSN 1548-7660. doi:10.18637/jss.v046.i10.

Isci S, Dogan H, Ozturk C, Otu HH (2014). "Bayesian Network Prior: Network Analysis of Biological Data Using External Knowledge." *Bioinformatics*, **30**(6), 860–867.

Jones DT, Buchan DW, Cozzetto D, Pontil M (2012). "PSICOV: Precise Structural Contact Prediction Using Sparse Inverse Covariance Estimation on Large Multiple Sequence Alignments." *Bioinformatics*, **28**(2), 184–190.

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.

Koller D, Friedman N (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lam W, Bacchus F (1994). "Learning Bayesian Belief Networks: An Approach Based on the MDL Principle." *Computational Intelligence*, **10**, 269–293. doi:10.1111/j.1467-8640.1994.tb00166.x.

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

Masegosa AR, Martínez AM, Ramos-López D, Cabañas R, Salmerón A, Nielsen TD, Langseth H, Madsen AL (2017). "**AMIDST**: A Java Toolbox for Scalable Probabilistic Machine Learning." *arXiv 1704.01427*, arXiv.org E-Print Archive. URL https://arxiv.org/abs/1704.01427.

Mazumder R, Friedman JH, Hastie T (2011). "SparseNet: Coordinate Descent with Nonconvex Penalties." *Journal of the American Statistical Association*, **106**(495), 1125–1138. doi:10.1198/jasa.2011.tm09738.

Meganck S, Leray P, Manderick B (2006). "Learning Causal Bayesian Networks from Observations and Experiments: A Decision Theoretic Approach." In *Modeling Decisions for Artificial Intelligence*, pp. 58–69. Springer-Verlag.

Meinshausen N, Bühlmann P (2006). "High-Dimensional Graphs and Variable Selection with the Lasso." *The Annals of Statistics*, **34**(3), 1436–1462. doi:10.1214/009053606000000281.

Murphy K (2014). "Software Packages for Graphical Models." Accessed 2019-10-31, URL https://www.cs.ubc.ca/~murphyk/Software/bnsoft.html.

Nicholson A, Cozman F, Velikova M, Van Scheltinga JT, Lucas PJF, Spaanderman M (2014). "Applications of Bayesian Networks Exploiting Causal Functional Relationships in Bayesian Network Modelling for Personalised Healthcare." *International Journal of Approximate Reasoning*, **55**(1), 59–73. doi:10.1016/j.ijar.2013.03.016.

Niinimäki T, Parviainen P, Koivisto M (2016). "Structure Discovery in Bayesian Networks by Sampling Partial Orders." *Journal of Machine Learning Research*, **17**(1), 2002–2048. doi:10.1007/978-3-642-23783-6_37.

Ono K, Muetze T, Kolishovski G, Shannon P, Demchak B (2015). "**CyREST**: Turbocharging Cytoscape Access for External Tools via a RESTful API." *F1000Research*, **4**(478). doi:10.12688/f1000research.6767.1.

Pearl J (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Peér D, Regev A, Elidan G, Friedman N (2001). "Inferring Subnetworks from Perturbed Expression Profiles." *Bioinformatics*, **17**, S215–S224.

Perrier E, Imoto S, Miyano S (2008). "Finding Optimal Bayesian Network Given a Super-Structure." *Journal of Machine Learning Research*, **9**(Oct), 2251–2286. doi:10.1007/978-3-642-25655-4_19.

Pournara I, Wernisch L (2004). "Reconstruction of Gene Networks Using Bayesian Learning and Manipulation Experiments." *Bioinformatics*, **20**(17), 2934–2942.

Ravikumar P, Wainwright MJ, Lafferty JD (2010). "High-Dimensional Ising Model Selection Using ℓ_1-Regularized Logistic Regression." *The Annals of Statistics*, **38**(3), 1287–1319. doi:10.1214/09-aos691.

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rosseel Y (2012). "**lavaan**: An R Package for Structural Equation Modeling." *Journal of Statistical Software*, **48**(1), 1–36. ISSN 1548-7660. doi:10.18637/jss.v048.i02.

Russell S, Norvig P (1995). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River.

Rütimann P, Bühlmann P (2009). "High Dimensional Sparse Covariance Estimation via Directed Acyclic Graphs." *Electronic Journal of Statistics*, **3**, 1133–1160. doi:10.1214/09-ejs534.

Sachs K, Perez O, Pe'er D, Lauffenburger DA, Nolan GP (2005). "Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data." *Science*, **308**(5721), 523–529. doi:10.1126/science.1105809.

Sanford AD, Moosa IA (2012). "A Bayesian Network Structure for Operational Risk Modelling in Structured Finance Operations." *Journal of the Operational Research Society*, **63**(4), 431–444. doi:10.1057/jors.2011.7.

Schmidt M, Niculescu-Mizil A, Murphy K (2007). "Learning Graphical Model Structure Using L1-Regularization Paths." In *AAAI*, volume 7, pp. 1278–1283.

Scutari M (2010). "Learning Bayesian Networks with the **bnlearn** R Package." *Journal of Statistical Software*, **35**(i03). doi:10.18637/jss.v035.i03.

Scutari M (2017). "Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimized Implementations in the **bnlearn** R Package." *Journal of Statistical Software*, **77**(2), 1–20. doi:10.18637/jss.v077.i02.

Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T (2003). "**Cytoscape**: A Software Environment for Integrated Models of Biomolecular Interaction Networks." *Genome Research*, **13**(11), 2498–2504. doi:10.1101/gr.1239303.

Shannon PT, Grimes M, Kutlu B, Bot JJ, Galas DJ (2013). "**RCytoscape**: Tools for Exploratory Network Analysis." *BMC Bioinformatics*, **14**(1), 217. doi:10.1186/1471-2105-14-217.

Spirtes P, Glymour C (1991). "An Algorithm for Fast Recovery of Sparse Causal Graphs." *Social Science Computer Review*, **9**(1), 62–72. doi:10.1177/089443939100900106.

Spirtes P, Glymour C, Scheines R (2000). *Causation, Prediction, and Search*, volume 81. The MIT Press.

Suzuki J (1993). "A Construction of Bayesian Networks from Databases Based on an MDL Principle." In *Proceedings of the Ninth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 266–273.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society B*, pp. 267–288. doi:10.1111/j.2517-6161.1996.tb02080.x.

Tsamardinos I, Brown LE, Aliferis CF (2006). "The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm." *Machine learning*, **65**(1), 31–78. doi:10.1007/s10994-006-6889-7.

Uhler C, Raskutti G, Bühlmann P, Yu B (2013). "Geometry of the Faithfulness Assumption in Causal Inference." *The Annals of Statistics*, **41**(2), 436–463. doi:10.1214/12-aos1080.

Van de Geer S, Bühlmann P (2013). "ℓ_0-Penalized Maximum Likelihood for Sparse Directed Acyclic Graphs." *The Annals of Statistics*, **41**(2), 536–567. doi:10.1214/13-aos1085.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wickham H, Hester J, Chang W, RStudio, R Core Team (2018). ***devtools**: Tools to Make Developing R Packages Easier*. R package version 2.0.1, URL https://CRAN.R-project.org/package=devtools.

Wu TT, Lange K (2008). "Coordinate Descent Algorithms for Lasso Penalized Regression." *The Annals of Applied Statistics*, pp. 224–244. doi:10.1214/07-aoas147.

Xiang J, Kim S (2013). "A* Lasso for Learning a Sparse Bayesian Network Structure for Continuous Variables." In *Advances in Neural Information Processing Systems*, pp. 2418–2426.

Yang E, Ravikumar P, Allen GI, Liu Z (2015). "Graphical Models via Univariate Exponential Family Distributions." *Journal of Machine Learning Research*, **16**, 3813–3847. doi:10.1016/b978-0-12-801522-3.00016-1.

Yuan M, Lin Y (2006). "Model Selection and Estimation in Regression with Grouped Variables." *Journal of the Royal Statistical Society B*, **68**(1), 49–67. doi:10.1111/j.1467-9868.2005.00532.x.

Zhang B, Gaiteri C, Bodea LG, Wang Z, McElwee J, Podtelezhnikov AA, Zhang C, Xie T, Tran L, Dobrin R, *et al.* (2013). "Integrated Systems Approach Identifies Genetic Nodes and Networks in Late-Onset Alzheimer's Disease." *Cell*, **153**(3), 707–720. doi:10.1016/j.cell.2013.03.030.

Zhang CH (2010). "Nearly Unbiased Variable Selection Under Minimax Concave Penalty." *The Annals of Statistics*, **38**(2), 894–942. doi:10.1214/09-aos729.

Zhang D (2016). *Concave Penalized Estimation of Causal Gaussian Networks with Intervention*. Master's thesis, UCLA Statistics 0891.

Zhang J, Spirtes P (2002). "Strong Faithfulness and Uniform Consistency in Causal Inference." In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pp. 632–639.

Zhou Q (2011). "Multi-Domain Sampling with Applications to Structural Inference of Bayesian Networks." *Journal of the American Statistical Association*, **106**, 1317–1330. doi:10.1198/jasa.2011.ap10346.

Affiliation:

Qing Zhou
UCLA Department of Statistics
Los Angeles, CA 90095, United States of America
E-mail: zhou@stat.ucla.edu
URL: http://www.stat.ucla.edu/~zhou/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2019, Volume 91, Issue 11 *Submitted:* 2017-03-07
doi:10.18637/jss.v091.i11 *Accepted:* 2018-03-08