Benchpress: A Scalable and Platform-Independent Workflow for Benchmarking Structure Learning Algorithms for Graphical Models

Felix L. Rios  
University of Basel

Giusi Moffa  
University of Basel

Jack Kuipers  
ETH Zürich

Abstract

Describing the relationship between the variables in a study domain and modelling the data generating mechanism is a fundamental problem in many empirical sciences. Probabilistic graphical models are one common approach to tackle the problem. Learning the graphical structure is computationally challenging and a fervent area of current research with a plethora of algorithms being developed. To facilitate the benchmarking of different methods, we present a novel automated workflow, called Benchpress for producing scalable, reproducible, and platform-independent benchmarks of structure learning algorithms for probabilistic graphical models. Benchpress is interfaced via a simple JSON-file, which makes it accessible for all users, while the code is designed in a fully modular fashion to enable researchers to contribute additional methodologies. Benchpress currently provides an interface to a large number of state-of-the-art algorithms from libraries such as BDgraph, BiDAG, bnlearn, GOBNILP, pcalg, r.blip, scikit-learn, TETRAD, and trilearn as well as a variety of methods for data generating models and performance evaluation. Alongside user-defined models and randomly generated data sets, the software tool also includes a number of standard datasets and graphical models from the literature, which may be included in a benchmarking workflow. We demonstrate the applicability of this workflow for learning Bayesian networks in four typical data scenarios. The source code and documentation is publicly available from http://github.com/felixleopoldo/benchpress.

Keywords: reproducibility, scalable benchmarking, probabilistic graphical models.

1. Introduction

Probabilistic graphical models play a central role in modern statistical data analysis. Their compact and elegant way to visualise and represent complex dependence structures in multivariate probability distributions have shown to be successfully applicable in many scientific domains, ranging from disciplines such as social sciences and image analysis to biology, medical diagnosis and epidemiology (see e.g. Elwert 2013; Friedman, Linial, Nachman, and Pe’er 2000; Friedman 2004; Moffa, Catone, Kuipers, Kuipers, Freeman, Marwaha, Lennox, Broome, and Bebbington 2017; Kuipers, Thurnherr, Moffa, Suter, Behr, Goosen, Christofori, and Beerenwinkel 2018b; Kuipers, Moffa, Kuipers, Freeman, and Bebbington 2019).

One of the main advantages of graphical models is that they provide a tool for experts and researchers from non-statistical fields to easily specify their assumptions in a specific problem.
domain, in a way that is very transparent. However, in many realistic situations the number of variables is either too large to build networks by hand, as it is for example the case in genomic applications, or simply no prior knowledge is available about the relationship between different variables. As a consequence, there has been a growing interest in automated strategies to infer the graph component of a probabilistic graphical model from data, so-called structure learning. For recent reviews of the wide flora of algorithms that have emerged in the last two decades, the reader is referred to Koski and Noble (2012); Constantinou, Liu, Chobtham, Guo, and Kitson (2021); Scutari, Graafland, and Gutiérrez (2019a).

Since the scientific principle of reproducibility is being recognised as instrumental to achieve high quality standards in research, there is an increasing demand to present new algorithms with publicly available source code and datasets (see e.g. Munafò, Nosek, Bishop, Button, Chambers, Percie du Sert, Simonsohn, Wagenmakers, Ware, and Ioannidis 2017; Lamprechts, Garcia, Kuzak, Martinez, Arcila, Martin Del Pico, Dominguez Del Angel, van de Sandt, Ison, Martinez, McQuilton, Valencia, Harrow, Psomopoulos, Gelpi, Chue Hong, Goble, and Capella-Gutierrez 2020). Ideally with publicly available software we should be able to easily compare different methods with respect to their performance in different settings. The reality is that the practical implementation of new approaches display high levels of heterogeneity on many aspects, which renders comparative studies both challenging and time consuming. One of the main difficulties is that different algorithms may be implemented in different programming languages, with different dependencies. Different packages may differ with respect to the representation of graphical models, the data formats and the way they interface to the user, e.g. through the command-line arguments, a configuration file or a function in a specific programming language. Another complication is that the number of computations increase rapidly in comparison studies, in particular when input parameters are altered, which requires parallel computation capabilities and careful bookkeeping to record and organise the results for reporting. As a result, researchers may invest considerable time on tailored benchmarking scripts that intend to address these issues on a subset of algorithms. To complicate matters beyond the implementational issues, there is no single well established metrics to evaluate the performance of structure learning algorithms. Quite the contrary there is a wide range of different metrics to choose from (see e.g. Tsamardinos, Brown, and Aliferis 2006; Constantinou et al. 2021) and researchers tend to use only a small subset to evaluate a new algorithm, typically with the aim of highlighting the problem their algorithm addresses.

With the problems described above in mind, the objective of this work is to develop a unified framework to facilitate execution and the benchmarking procedure of different algorithms. We present a novel workflow called benchpress, which is implemented in snakemake (Köster and Rahmann 2012) and designed for making reproducible, platform-independent, and fully scalable benchmarks of structure learning algorithms. Benchpress is interfaced via an easy-to-use configuration file in JSON-format (Pezoa, Reutter, Suarez, Ugarte, and Vrgoč 2016), where a benchmark setup is specified in a module based probabilistic programming style, separating model specification, in terms of graph and corresponding parameters, from algorithm execution and evaluation. Snakemake enables benchpress to seamlessly scale the computations to server, cluster, grid and cloud environments, without any changes in the configuration file. The support for containerized software through e.g. Singularity (Kurtzer, Sochat, and Bauer 2017) or docker (Merkel 2014) images, together with platform-independent representations of data sets and graphs enables benchpress to compare algorithms from different libraries, possibly implemented in different programming languages with different dependencies.
Benchpress is implemented in a modular coding style which facilitated for researchers to contribute with additional modules for generating models, structure learning algorithms and evaluating results. In the first publicly released version we already included 19 algorithms from some of the most popular libraries such as BDgraph (Mohammadi and Wit 2019), BiDAG (Suter, Kuipers, Moffa, and Beerenwinkel 2021), bnlearn (Scutari 2010), GOBNILP (Cussens 2020), pcalg (Kalisch, Mächler, Colombo, Maathuis, and Bühlmann 2012), r.blip (Scanga, de Campos, Corani, and Zaffalon 2015), seikit-learn (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg et al. 2011), TETRAD (Glymour and Scheines 1986), and trilearn (Olsson, Pavlenko, and Rios 2019) along with some of the models and data sets from the standard literature.

Recently Constantinou, Liu, Chobtham, Guo, and Kitson (2020) developed Bayesys, a software with similar intentions, albeit exclusively designed for Bayesian networks, a specific type of probabilistic graphical models (Pearl 1997). Furthermore Bayesys is a Java implementation only including one algorithm for structure learning, called SaiyanH (Constantinou 2020), suited for high dimensional settings. Another tool called causal-compare with similar purpose as benchpress was recently released as part of the TETRAD project (Ramsey, Malinsky, and Bui 2020). However, the functionality of causal-compare is restricted to the algorithm implementations in the TETRAD project, which represents only a subset of available algorithms (Ramsey et al. 2020). This is makes its use more limited compared to benchpress, which only requires a Linux system while there is no limitation for the individual algorithms requirements.

Benchpress is distributed under the GPL v2.0 License, encouraging its use in both academic and commercial settings. Source code and documentation are available from https://github.com/felixleopoldo/benchpress.

The rest of the paper is structured as follows. Section 2 presents an introduction to graphical models along with some notational conventions. Section 3 provides background on current strategies for structure learning. Section 4 describes the structure of the JSON interface and the modules already available for benchmarking. In Section 5, we show how to use benchpress in four typical benchmarking scenarios. Installation and usage guidelines are provided in Section 6.

2. Probabilistic graphical models

This section provides a short introduction to graphical models and graph theory, with definitions as in e.g. Diestel (2005); Lauritzen (1996).

Let $P$ be a $p$-dimensional distribution and let $Y := (Y_i)_{i=1}^p$ be a random vector such that $Y \sim P$. Further let $G = (V, E)$ be a graph, where $V = \{Y_i\}_{i=1}^p$ is the node set and $E$ is the edge set, consisting of tuples of distinct elements of $V$. Then $P$, or alternatively $Y$, is said to be Markov with respect to $G$ if separation, based on some graphical criteria, of two subsets of nodes, $Y_A$ and $Y_B$, by a third set $Y_C$ in $G$ implies their conditional independence in $P$, where $Y_A := (Y_i)_{i \in A}$. The term probabilistic graphical model or graphical model has been used interchangeably in the literature to refer to either the tuple $(G, P)$ or a collection of distributions $P_G$, usually restricted to some specific parametric family, where every $P' \in P_G$ is Markov with respect to $G$. In this text we will use the former meaning. In the case where $P$ is restricted to a specific parametric family, it is usually uniquely determined by some parameter.
vector $\Theta$, thus either notation may be used.

Assuming $f$ is a density for $P$, conditional independence between the random variables $Y_A$ and $Y_B$ given $Y_C$ is well-defined as $f(y_A|y_B, y_C) = f(y_A|y_C)$, where $(y_A, y_B, y_C) \in Y_A \times Y_B \times Y_C$ and $Y_A$ denotes the range of $Y_A$. In contrast, node separation may have different definitions depending on the type of the graph considered. Typical classes of graphs are: 

- **undirected graphs**, where edges are specified as unordered tuples and separation of $Y_A$ and $Y_B$ given $Y_C$ means that every path between $Y_A$ and $Y_B$ must pass $Y_C$;

- **directed acyclic graphs (DAGs)**, where edges are represented as ordered tuples and separation is defined by means of a concept known as *d*-separation, see e.g. Pearl (1997).

Next follows a short introduction to the type of graphical models currently implemented in *benchpress*.

**Bayesian networks** constitute one of the most popular classes of graphical models, referring to distributions that are Markov with respect to a DAG. The conditional independence encoded by the DAG implies that the density can be factorised into local conditional densities, where each node $Y_i$ is directly dependent on its parents, $pa(Y_i)$, as

$$f(y) = \prod_{i=1}^{p} f(y_i|pa(y_i)),$$

where $f(y_i|pa(y_i))$ denotes the conditional density of $Y_i$ given $pa(Y_i) = pa(y_i)$. This property is appealing for several reasons. Firstly, the partition into local conditional probability distribution provides an intuitive way to express and visualise knowledge about a specific problem domain. In particular, an expert could express a causal mechanism in terms of a DAG, which in turn provides a mathematical language to answer causal queries from non-experimental data (Pearl 1995). Further, it enables fast computations of conditional and marginal distribution using message-passing algorithms by exploiting the factorisation property in the distribution. However, when the DAG is not used for expressing causal knowledge, the direction of the edges may be misleading since the conditional independencies of a distribution are not uniquely encoded by a single DAG in general but rather by a class of *Markov equivalent* DAGs, all of which encode the same conditional independence statements. One way to represent a Markov equivalence class is by a *completed partially directed acyclic graph* (CPDAG, see Chickering 1995). A CPDAG is a partially directed DAG which can be obtained from a DAG by regarding an edge $(Y_i, Y_j)$ as undirected if and only if both directed edges $(Y_i, Y_j)$ and $(Y_j, Y_i)$ are present in some of the DAGs in the same equivalence class. The CPDAG is also sometimes called the *essential graph*.

**Markov networks** or **Markov random fields** are another popular class of graphical models, referring to distributions that are Markov with respect to an undirected graph. There is in general no decomposition into local densities for the models in this class. However, there exist functions $\{\phi_C\}_{C \in C}$ such that

$$f(y) = \prod_{C \in C} \phi_C(y_C),$$

where $C$ is the set of maximal subgraphs where all nodes are directly connected with each other, usually called *cliques*, which may be leveraged to calculate marginal distributions efficiently. There is a sub-class of undirected graphs called *decomposable* (DG) or *chordal* or *triangulated*, where each cycle with more than three nodes has a *chord*, an edge which breaks it. Distributions which are Markov with respect to decomposable graphs are called *decomposable*
and their densities are built by local, clique-specific densities, \( \{ f_C \}_{C \in \mathcal{C}} \) and their marginals so that the joint density factorises as

\[
f(y) = \prod_{C \in \mathcal{C}} f_C(y_C) \prod_{S \in \mathcal{S}} f_S(y_S),
\]

where \( \mathcal{S} \) is the multi-set of separators in \( G \), found \textit{e.g.} as the intersection of neighboring cliques in a so-called \textit{junction tree}, built from the cliques in the graph; see \textit{e.g.} Lauritzen (1996) for details. In general Markov networks and Bayesian networks encode different sets of conditional independence statements, and there are sets which we can represent with one model but not the other. Decomposable graphs represent the intersection of those sets of statements which we can represent by both undirected graphs and DAGs (see \textit{e.g.} Lauritzen 1996; Cowell, Dawid, Lauritzen, and Spiegelhalter 2003; Koller and Friedman 2009). The DAGs representing the intersection set are called \textit{perfect}, meaning that all the parents of each node are connected by an edge. Further characterisations of decomposable graphs are found in \textit{e.g.} Lauritzen (1996); Cowell et al. (2003) and Duarte and Solus (2021).

3. Structure learning

Structure learning algorithms for Bayesian networks was proved to be NP-hard, even when the number of parents of each node is restricted to two (Chickering, Heckerman, and Meek 2004). Also the number of undirected graphs grows as \( O(2^p^2) \) even when restricted to decomposable graphs, making an exhaustive search for a specific graph infeasible (see \textit{e.g.} Wormald 1985; Olsson, Pavlenko, and Rios 2018). Therefore, in practice we must rely on approximation methods which we can divide into three main categories: \textit{constraint-based}, \textit{score-based}, and \textit{hybrid} algorithms, discussed briefly below.

The seed to the first constraint-based algorithm was planted by Verma and Pearl (1991) in the context of causal Bayesian networks. In line with their work, constraint-based methods employ conditional independence tests among the variables to first estimate an undirected graph \textit{by, e.g.}, starting with the complete graph, and excluding an edge \((Y_i, Y_j)\) if there exists some set \( Y_A \) such that the hypothesis \( Y_i \perp \!\!\!\!\perp Y_j \mid Y_A \) can be not rejected according to a suitable test procedure. Such methods tend to be very fast but testing can suffer from large numbers of false negatives, \textit{i.e.} falsely not including edges when they should be present. Relaxing the significance level of the tests to reduce false negatives can however rapidly inflate false positives and runtimes.

Score-based methods on the other hand aim to optimise a global score function defined on the graph space. As a work around to cope with the immense number of graphs, the search is often limited to certain types of graphs. For example in DAG models, one may restrict the number of parents or combine DAGs into bigger and more easily represented classes (\textit{e.g.} orders and partitions Friedman and Koller 2003; Kuipers and Moffa 2017). For decomposable graphs a possibility is to limit the maximal clique size. Score functions may rely on penalised log-likelihoods or on Bayesian posterior probabilities of graphical structures conditional on the observed data (see \textit{e.g.} Koller and Friedman 2009). Besides mere optimisation, Bayesian structure learning also focuses on sampling procedure to provide a finer characterisation of the posterior graph distribution. When an expression for the posterior distribution is available at least up to a normalising constant, we can implement Markov chain Monte Carlo (MCMC) schemes to sample from it, see \textit{e.g.} Madigan, York, and Allard (1995); Giudici and Castelo.
Benchpress: a scalable and platform-independent workflow for structure learning

(2003); Friedman and Koller (2003) for early works and Kuipers and Moffa (2017); Jennings and Corcoran (2018) for more recent strategies. In Gaussian Markov networks, Monte Carlo sampling turns out to be infeasible for larger network since there is in general no analytic expression for the un-normalised posterior (Atay-Kayis and Massam 2005). On the other hand, score-based methods such as graphical lasso, relies on the fact that the edges in the graph are revealed from the non-zero pattern of the elements in the precision (inverse covariance) matrix. Optimisation techniques can successfully estimate the \( L_1 \)-penalised log-likelihood of the precision matrix, and the graph as a by-product; see Friedman, Hastie, and Tibshirani (2008); Meinshausen (2008); Witten, Friedman, and Simon (2011). Decomposable models, do possess a closed form expression for the un-normalised graph posterior and most inferential methods built on the representation via undirected graphs focus on Monte Carlo sampling (see e.g. Giudici and Green 1999; Green and Thomas 2013; Elmasri 2017; Olsson et al. 2019) with a few focusing on optimisation (see e.g. Carvalho 2006; Studený and Cussens 2017; Rantanen, Hyttinen, Järvisalo et al. 2017).

Hybrid algorithms typically use constraint-based algorithms as a preliminary step called restrict phase in order to reduce the search space for a score-based method. A score based-algorithm is then used in the maximise phase (see e.g. Tsamardinos, Aliferis, Statnikov, and Statnikov 2003; Scanagatta et al. 2015; Kuipers, Suter, and Moffa 2018a).

4. The JSON configuration file and supported modules

In this section we describe the structure of the JSON (Pezoa et al. 2016) configuration file which serves as interface to benchpress. The JSON file provides a simple key-value format which is easy to interpret and modify. We will use the typewriter style to indicate keys and bold italic style for values. When the value of a key is another JSON object or a list, we will sometimes refer to it as a section. At the highest level there are two main sections, benchmark_setup and resources.

The resources section contains separate subsections of the available modules for generating or defining graphs (graph), parameters (parameters), data (data), and algorithms for structure learning (structure_learning_algorithms). Each module in turn has a list, where each of its elements are objects defining a parameter setting, identified with a unique id. See Listings 1 for an example object in the pcalg_pc module. The benchmark_setup section specifies the data models (data) and evaluation methods (evaluation) to be considered for analysis. The module objects used in benchmark_setup are those defined in resources and referenced by their corresponding id’s. The output of each module is saved systematically under the results/ directory based on the corresponding objects values. A more detailed description of the structure of each section and how to use the modules supported so far follows in the next section.

It is important to note that one of the main strengths of benchpress is that the modular design provides a flexible framework to integrate new modules into any of the sections. To do so, one essentially needs either a docker (or singularity) image or a local installation, where the new module is installed and obeys the input and output formats specified in Appendix A. The snakemake workflow should then be adapted for the new module following the same structure as the existing modules in its section. The reader is referred to Boettiger and Eddelbuettel (2017); Nüst, Sochat, Marwick, Eglen, Head, Hirst, and Evans (2020) for a comprehensive
introduction to docker in data science.

4.1. Benchmark setup (benchmark_setup)

Data setup section (data)

Benchpress supports different strategies for defining models and data. One possibility is to generate data, graphs and parameters using the existing modules, referenced by their objects id, as described above. An alternative is to provide user-specified graphs, parameters and data sets, referenced by their corresponding filenames. Further we can combine the two approaches as needed.

The data section consists of a list of objects, each containing the id of a graph (graph_id), parameters (parameters_id), data (data_id) module object, and a tuple (seed_range), specifying the range of the random seeds used in the modules. In case of fixed data sets, parameters, or graphs, we simply use the corresponding filenames in place of id’s.

For each seed number i in the range defined by seed_range, a graph Gi is obtained as specified according to graph_id. Given Gi, the parameters in the model Θi are obtained according to the specifics in parameters_id. A data matrix, YT i = (Yj1:p)n j=1, is then sampled from (Gi,Θi) as specified by the data model in data_id. More details and examples are given later in Section 4.2.

Evaluation section (evaluation)

There is typically no algorithm which performs well with respect to all performance metrics, and no single metric which is generally preferred. Therefore, to get an overall picture of the performance of an algorithm benchpress supports different metrics. This section describes the metrics implemented in benchpress, where we let G = (V,E) and G′ = (V ′, E′) denote the true and the estimated graph respectively.

The first choice is the space on which to compare the true and inferred structures. The currently supported Markov-equivalent types are CPDAGs, pattern graphs and the skeleton of the graphs (undirected version of a graph). The pattern graph of a DAG is the graph obtained by keeping the directions of the v-structures, and regarding the remaining edges as undirected. v-structures are characterised by triples of nodes where the edges of two of the nodes, which should not be neighbors, are pointing to the third. Within a given comparison space, the evaluation section below contains a list of the currently available evaluation modules.

Structural Hamming distance

The structural Hamming distance (SHD) is one of the most commonly used metrics to compare graphs. It describes the number of changes, in terms of adding, removing or reversing edges, needed so that E = E′.

Receiver operating characteristic (roc)

The roc module defines a metric for comparing graphs which may also be applied to mixed graphs, having a combination of edges oriented in different ways (directed or undirected), thus we distinguish between the orientations.
We assign to every edge \( e \in E' \) the true positive score \( TP(e) = 1 \) if \( e \) is contained in \( E \) with the same orientation (\( e \) being undirected in both \( E \) and \( E' \) or \( e \) having the same direction in both \( E \) and \( E' \)), \( TP(e) = 1/2 \) if \( e \) is contained in \( E \) with a different orientation, otherwise \( TP(e) = 0 \). The false positive score \( FP(e) \) is defined analogously. First let \( \bar{E} \) denote the complementary set of \( E \) in the sense that directed edges are reversed and undirected edges (non-edge) are removed (added). For every edge \( e \in \bar{E} \), \( FP(e) = 1 \) if \( e \) is contained in \( \bar{E} \) with the same orientation and \( FP(e) = 1/2 \) if \( e \) is contained in \( \bar{E} \) with a different orientation, otherwise \( FP(e) = 0 \).

The total true and false positive edges (TP and FP) are obtained as the sum of the individual edge scores, i.e.

\[
TP := \sum_{e \in E} TP(e), \quad FP := \sum_{e \in E} FP(e).
\]

We define the true and false positive rates (FPRp and TPR) as

\[
FPRp := \frac{FP}{P}, \quad TPR := \frac{TP}{P},
\]

where \( P := |E| \) denotes the total number of edges in \( G \). Note that TP and FP reduces to the ordinary counting true and false positives when all edges are undirected in both \( G \) and \( G' \).

The roc module evaluates the TPR and FPRp for each seed in \( \text{seed\_range} \) and the median (mean) FPRp is plotted against the median (mean) TPR in ROC type sub figures using \texttt{ggplot2} (Wickham, Averick, Bryan, Chang, McGowan, François, Grollemund, Hayes, Henry, Hester, Kuhn, Pedersen, Miller, Bache, Müller, Ooms, Robinson, Seidel, Spinu, Takahashi, Vaughan, Wilke, Woo, and Yutani 2019) based on the graph, parameters and data modules used. The top three title rows of each sub figure describe, from top to bottom, the objects from the graph, parameters and data modules. The varying parameter forming the curve should be given as a list of values in the corresponding algorithm object. A parameter could for example be the significance level in a test procedure of a constraint-based algorithm, such as \texttt{alpha} in Listing 1, or a parameter of the score function in a score-based method.

The id’s for the objects that should be part of the evaluation should be placed in a list called \texttt{ids}. The \texttt{filename\_prefix} field specifies a prefix of the produced files. The user can choose to show points (points), parameter values (text), lines between the points or values (path), and error bars showing the 5\% and 95\% quantiles of the TPR (errorbar).

There is an interesting connection between the SHD metric and the ROC curve when TPR and FPRp are defined as above. In such (possibly mixed) graphs, the SHD can easily be determined in a ROC figure as the scaled \textit{Manhattan distance} between the points \((0,1)\) and \((TPR, FPRp)\), i.e.

\[
SHD/P = 1 - TPR + FPRp.
\]

Consequently, the ROC curves enable the simultaneous analysis of SHD, TPR, and FPRp in the same figure.

The benchmarks are saved in \texttt{.csv} files in the directory \texttt{results/benchmarks/}. A summary of the benchmarks for each algorithm is saved directly in \texttt{results/output} according to the algorithm’s module name and the value of the \texttt{filename\_prefix} field. For example, if \texttt{filename\_prefix}
is set to myplots/ the results of the tetrad_gfci module are stored in results/output/myplots/tetrad_gfci.csv. The plots are saved under results/output/myplots/ along with the source file results/output/myplots/ROC_data.csv.

Plot true adjacency matrix (adjmat_true_plots)
The adjmat_true_plots module plots the adjacency matrices of the true graphs. The plots are saved in sub directories of results/adjmat/ and copied to results/output/adjmat_true_plots/ for easy reference.

Plot adjacency matrix (adjmat_plots)
The adjmat_plots module plots the adjacency matrices of the estimated graphs. The figures are saved in results/adjmat and copied to results/output/adjmat_plots/.

Plot estimated graphs (graph_plots)
The graph_plots module plots and saves the estimated graphs in .dot format. The figures are saved in results/adjmat and copied to results/output/graph_plots/.

Plot true graphs (graph_true_plots)
The graph_true_plots module plots the true underlying graphs. The figures are saved in results/adjmat and copied to results/output/graph_true_plots/.

MCMC mean graphs (mcmc_heatmaps)
As mentioned in Section 3, for Bayesian inference it is custom to use MCMC methods to simulate a Markov chain of graphs \( \{G^l\}_{l=0}^\infty \) having the graph posterior as stationary distribution. Suppose we have a realisation of length \( M + 1 \) of such chain, then the posterior edge probability of an edge \( e \) is estimated by

\[
\frac{1}{M+1} \sum_{l=0}^M 1_{e}(e^l),
\]

where the first samples \( b \) are disregarded as a burn-in period.

The mcmc_heatmaps module has a list of objects, where each object has an id field for the algorithm object id and a burn-in field (burn_in) for specifying the burn-in period. The estimated probabilities are plotted in heatmaps using seaborn (Waskom 2021) which are saved in results/mcmc_heatmaps/ and copied to results/output/mcmc_heatmaps/ for easy reference.

MCMC trajectory plots (mcmc_traj_plots)
The mcmc_traj_plots module plots the so called score values in the trajectory or the value of a given functional. The currently supported functionals are the number of edges for the graphs (size) and the graph score. The mcmc_traj_plots module has a list of objects, where each object has an id field for the algorithm object id, a burn-in field (burn_in) and a field specifying the functional to be considered (functional). Since the trajectories tend to be very long, the user may choose to thin out the trajectory by only considering every graph at a given interval length specified by the thinning field. The plots are saved in results/mcmc_traj_plots/ and copied to results/output/mcmc_traj_plots/.

MCMC auto-correlation plots (mcmc_autocorr)
The \texttt{mcmc_autocorr} module plots the auto-correlation of a functional of the graphs in a MCMC trajectory. Similar to the \texttt{mcmc_traj_plots} module, the \texttt{mcmc_autocorr} module has a list of objects, where each object has an \texttt{id}, \texttt{burn_in}, \texttt{thinning}, and a \texttt{functional} field. The maximum number of lags after thinning, is specified by the \texttt{lags} field. The plots are saved in \texttt{results/mcmc_autocorr/} and copied to \texttt{results/output/mcmc_autocorr/}.

4.2. Resources section

\textit{Graph, G (graph)}

Several possibilities exist for defining the graph component, depending on whether we wish to draw random graphs from a given distribution, or maybe provide a given structure on which we wish to perform a benchmarking study. Below is the list of currently available modules. The adjacency matrices of the generated graphs are saved in the directory \texttt{results/adjmat/}, where each generation method has its own sub directory, named by their module name and structured by their objects values.

\textit{Random DAG with constrained number of parents per node (pcalg_randdag)}

An object of the \texttt{pcalg_randdag} module specifies a graph generated using the \texttt{randDAG} function from the \texttt{pcalg} package (Kalisch \textit{et al.} 2012), with the extra functionality of restricting the maximal number of parents per node (\texttt{max_parents}). It samples a random graph with \texttt{n} nodes and \texttt{d} neighbours per node on average, using the algorithm specified by \texttt{method} with parameters \texttt{par1} and \texttt{par2}, for further details see Kalisch \textit{et al.} (2012).

\textit{Random undirected graph (bdgraph_graphsim)}

An object of the \texttt{bdgraph_graphsim} module specifies a graph generated using the \texttt{graph.sim} function from the \texttt{BDgraph} package (Mohammadi and Wit 2019).

\textit{Band matrix (bandmat)}

An object of the \texttt{bandmat} module specifies a decomposable graph with adjacency matrix of given bandwidth (\texttt{bandwith}).

\textit{Band matrix with random width (rand_bandmat)}

An object of the \texttt{max_bandwith} module specifies a decomposable graph by randomly sampling a band structured adjacency matrix of a given maximum width (\texttt{max_bandwith}).

\textit{Christmas tree algorithm (trilearn_cta)}

An object of the \texttt{trilearn_cta} module defines decomposable graph randomly generated using the \texttt{Christmas tree algorithm} (Olsson \textit{et al.} 2018), with the sparsity parameters \texttt{alpha} and \texttt{beta}.

\textit{Fixed adjacency matrix}
A fixed graph is represented as an adjacency matrix and should be formatted as specified in Appendix A. The file should be stored with the .csv extension in the directory resources/adjmat/myadjmats/.

Model parameters, $\Theta$ (parameters)

Depending on how the user chooses to provide data and graph there are different possibilities for setting the model parameters. As for the graph module, one possibility is to define parameters is by a filename. Another is to use one of the modules listed below. In any case, it is assumed that the underlying graph $G$ is the one specified by graph_id in the data section. The model parameters are saved in the directory results/parameters/, where each generation method has its own sub directory named by their module and structured by the object values and corresponding graph.

Random binary Bayesian network (bin_bn)

Provided that $G$ is a DAG, an object of the bin_bn module defines a binary Bayesian network (where the nodes are binary variables) by sampling its conditional probability tables. More specifically, the conditional probability tables for each variable $Y_i$ and parent configuration $\text{pa}(Y_i)$ are distributed as

$$\text{pr}(Y_i = 0|\text{pa}(Y_i)) \sim \text{Unif}([a, b]),$$

where $(a, b) \in [0, 1]^2, a < b$ and Unif($c$) denotes the uniform distribution on the set $c$.

Using an id of this module in the parameters_id field of the data section requires that graph_id represents a DAG (DAG module object or DAG adjacency matrix).

Random linear Gaussian Bayesian network (sem_params)

Provided that $G$ is a DAG, an object of the sem_params module defines a Bayesian network by generating the weight matrix $W$ of a Gaussian linear structural equation model (SEM) of the form

$$Y_i = \sum_{j: Y_j \in \text{pa}(Y_i)} W_{ij}Y_j + Z_i,$$

where $Z_i \sim \mathcal{N}(\mu, \sigma^2)$ and elements of $W$ are distributed as

$$W_{ij} \sim \begin{cases} \text{Unif}([a, b])\text{Unif}([-1, 1]) & \text{if } (i, j) \in E \\ 0 & \text{otherwise}. \end{cases}$$

Using an object id of this module in the parameters_id field of the data section requires that graph_id represents a DAG.

Fixed Bayesian network

A Bayesian network object in the R-package bnlearn (Scutari 2010) is an instance of the bn.fit class and contains both the DAG and corresponding parameters. Motivated by the modular design of benchpress, a bn.fit object may be used to specify only the parameters of a Bayesian
network, by storing it in .rds format in resources/parameters/myparms/bn_fit_networks/. The adjacency matrix of the DAG should be stored in the folder for graphs, as specified in Section 4.2.1 and graph_id should be set to the filename. Benchpress already contains a number of standard Bayesian network models, e.g. those available at the website of bnlearn, including the so-called Bayesian network repository (UAI) (Friedman, Goldszmidt, Heckerman, and Russell 1997).

A matrix defining the weights $W$ in a SEM as defined in (1) can be stored in resources/parameters/myparms/sem_params/ with the .csv extension, formatted similarly to the adjacency matrix specified in Appendix A. This requires that graph_id represents a DAG.

Gaussian graph intra-class model (intraclass)
Provided that $G$ is undirected, an object of the intraclass module defines a zero mean multivariate Gaussian distribution by its covariance matrix $\Sigma$ as

$$\Sigma_{ij} = \begin{cases} \sigma^2, & \text{if } i = j \\ \rho \sigma^2, & \text{if } (i, j) \in E \end{cases}$$

and $(\Sigma^{-1})_{ij} = 0$ if $(i, j) \notin E$, where $\sigma^2 > 0$ (sigma2) and $\rho \in [0, 1]$ (rho) denote the variance and correlation coefficient, respectively.

Using an object id of this module in the parameters_id field of the data section requires that graph_id represents an undirected graph.

Hyper Dirichlet (trilearn_hyper-dir)
Provided that $G$ is decomposable, an object of the trilearn_hyper-dir module defines a categorical decomposable model by sampling the parameters from the hyper Dirichlet distribution (Dawid and Lauritzen 1993), with specified concentration parameter (pseudo_obs) and number of levels per variable (n_levels).

Hyper inverse-Wishart (trilearn_hyper_inv_wishart)
Provided that $G$ is decomposable, an object of the trilearn_hyper_inv_wishart module defines a zero-mean Gaussian decomposable model by sampling the covariance matrix from the Hyper inverse-Wishart distribution (Dawid and Lauritzen 1993) according to the procedure outlined by Carvalho, Massam, and West (2007). The clique-wise scale matrices are fixed to the identity matrix and the degrees of freedom subtracted by $p$ (dof) should be specified by the user.

Inverse $G$-Wishart (bdgraph_rgwish)
Provided that $G$ is undirected, an object of the bdgraph_rgwish module defines a zero-mean Gaussian graphical model by sampling the precision matrix from the $G$-Wishart distribution (Atay-Kayis and Massam 2005; Lenkoski 2013) using the rgwish function from the BDgraph package (Mohammadi and Wit 2019). The clique-wise scale matrices are fixed to the identity matrix and the degrees of freedom subtracted by $p$ (b) and a threshold (threshold) value for the convergence of sampling algorithm should be specified by the user. Benchpress saves the inverted precision matrix, serving as covariance matrix in a Gaussian distribution.
Data, Y (data)

There are two possibilities for the data section. The user may provide their own data in .csv format for benchmarking on real data. Alternatively benchpress can generate data according to a data model for benchmarking on simulated data. The data sets are saved in results/data/, structured according to the generation modules of the graph and parameters and their object values or filenames.

Fixed data sets

The are two ways of providing fixed data sets. The first option is to place data files directly in the directory resources/data/mydatasets/ with the .csv extension, formatted as specified in Appendix A. The second option is to place the files in a sub directory of resources/data/mydatasets/. In the latter case, the data_id field should be the name of the directory and all the files in it will be considered for evaluation.

Note that, when a filename is used as value for data_id, since the data is fixed, seed_range should be set to null. However, depending on the evaluation method used, the graph_id could be either null or the filename of the true graph underlying the data. For example, the roc module requires that graph_id set to a proper id for a graph object, however it is not a requirement for the adjmat_plots module. When the value of data_id is a directory, the other fields should be set to null.

Independent identically distributed (i.i.d.) samples (iid)

An object of the iid module will draw a specified (sample_sizes) number of independent samples according to the module id’s or filenames specified in graph_id and parameters_id. Continuous data may be standardized by setting standardized to true. See Reisach, Seiler, and Weichwald (2021) for a discussion about standardising data in a structure learning context.

Structure learning algorithms (structure_learning_algorithms)

This section contains a short summary, sorted in a temporal ordering based on publication year, of the currently 19 structure learning modules supported by benchpress that are also used in the simulation studies in Section 5. See Table 1 for a summary of their properties and implementations. The full list of available algorithms is found in the documentation of benchpress.

PC (pcalg_pc)

The PC algorithm (Spirtes and Glymour 1991), is a constraint based method consisting of two main steps. The first step is called the adjacency search and amounts to finding the undirected skeleton of the DAG. The second step amounts to estimating a CPDAG. See Listing 1 for an example.

Fast causal inference (tetrad_fci)

Fast causal inference (FCI) is a two-stage algorithm similar to the PC algorithm, where the
first step finds the skeleton and the second step orients the edges in a DAG (Spirtes, Glymour, Scheines, and Heckerman 2000). The FCI is designed for models with latent variables.

**Hill climbing (bnlearn_hc)**

Hill climbing (HC) is a score-based algorithm which starts with a DAG with no edges and adds, deletes or reverses edges in a greedy manner until an optimum is reached (Russell and Norvig 2002; Scutari, Vitolo, and Tucker 2019b).

**Tabu (bnlearn_tabu)**

Tabu is a less greedy version of the HC algorithm allowing for non-optimal moves that might be beneficial from a global perspective to avoid local maxima (Russell and Norvig 2002; Scutari et al. 2019b).

**Really fast causal inference (tetrad_rfci)**

Really fast causal inference (RFCI) is a modified version of the the FCI algorithm, where some additional tests before orienting v-structures and discriminating paths are introduced, making it much faster, in particular for sparse graphs in high-dimension (Colombo, Maathuis, Kalisch, and Richardson 2012).

**Fast greedy equivalent search (tetrad_fges)**

Fast greedy equivalent search (FGES) is a score based method based on the the greedy equivalence search (GES) of Meek (1997). This algorithm operates on the space of CPDAG’s (Chickering 2002). Its complexity is polynomial in the number of nodes.

**Greedy fast causal inference (tetrad_gfci)**
The FGES is asymptotically correct under the unrealistic assumption that there are no unmeasured confounders (Ogarrio, Spirtes, and Ramsey 2016). However, FCI and RFCI often perform poorly in scenarios with small sample sizes. Greedy fast causal inference (GFCI) is a hybrid method combining elements from the FCI and FGES algorithms, while making no assumptions about unmeasured confounders. It is expected to perform better on small samples.

**Order Markov chain Monte Carlo (bidag_order_mcmc)**

This technique relies on a Bayesian perspective on structure learning, where the score of a DAG is defined as its posterior distribution. To overcome the limitation of simple structure-based MCMC schemes, Friedman and Koller (2003) turned to a score defined as the sum of the posterior scores of all DAG which are consistent with a given topological ordering of the nodes. One can then run a Metropolis-Hasting algorithm to sample from the distribution induced by the order score, and later draw a DAG consistent with the order. This strategy substantially improves convergence with respect to earlier structure MCMC scheme, though it unfortunately produces a biased sample on the space of DAGs. The implementation considered in benchpress is a hybrid version with the sampling performed on a restricted search space initialised with constraint-based testing and improved with score-based search (Kuipers et al. 2018a).

**Grow-shrink (bnlearn_gs)**

The grow-shrink (GS) algorithm is based on the Markov blanket of the nodes in a DAG. For a specific node, the Markov blanket it the set of nodes which conditioning upon renders it conditionally independent from all other variables (Margaritis 2003). It is a constraint-based method which estimates the Markov blanket of a node in a two-stage forward-backward procedure using conditional independence tests. The Markov blankets are used to first estimate an undirected graph and then estimate a DAG in a four-step procedure.

**Incremental association Markov blanket (bnlearn_interiamb)**

As in GS, this algorithm is also based on the Markov blanket method to first determine the undirected skeleton. However, in incremental association Markov blanket (inter-IAMB) the variable to be included in the Markov blankets are not considered in static order as in GS and the forward-backward stages are combined into a single procedure, which has the effect of reducing the size of the blankets (Tsamardinos et al. 2003).

**Max-min hill-climbing (bnlearn_mmhc)**

Max-min hill-climbing (MMHC) is a hybrid method which first estimates the skeleton of a DAG using an algorithm called Max-Min Parents and Children and then performs a greedy hill-climbing search to orient the edges with respect to a Bayesian score (Tsamardinos et al. 2006). It is a popular approach used as standard benchmark and also well suited for high-dimensional domains.

**Globally optimal Bayesian network learning using integer linear programming (gobnilp)**
A score-based method using integer linear programming (ILP) for learning an optimal DAG for a Bayesian network with limit on the maximal number of parents for each node (Cussens 2012). It is a two-stage approach where candidate parent sets for each node are discovered in the first phase and the optimal sets are determined in a second phase.

**Acyclic selection ordering-based search** *(rblip_asobs)*

A score-based two-phase algorithm where the first phase aims to identify the possible parent sets, Scanagatta et al. (2015); Scanagatta, Corani, De Campos, and Zaffalon (2018). The second phase performs an optimisation on a modification of the space of node orders introduced in Teyssier and Koller (2012), allowing edges from nodes of higher to lower orders, provided that no cycles are introduced.

**Iterative search** *(bidag_itsearch)*

This is a hybrid score-based optimisation technique based on Markov chain Monte Carlo schemes (Kuipers et al. 2018a). The algorithm starts from a skeleton obtained through a fast method (e.g. a constraint-based based method, or GES). Then it performs score and search on the DAGs belonging to the space defined by the starting skeleton. To correct for edges which may be missed, the search space is iteratively expanded to include one additional parent for each variable from outside the current search space. The score and search phase relies on an MCMC scheme producing a chain of DAGs from their posterior probability given the data.

**No tears** *(notears)*

This score-based method recasts the combinatorial problem of estimating a DAG into a purely continuous non-convex optimization problem over real matrices with a smooth constraint to ensure acyclicity (Zheng, Aragam, Ravikumar, and Xing 2018).

5. **Simulation study**

The following subsections demonstrate four data scenarios to benchmark structure learning algorithms for Bayesian networks; where we consider binary and continuous data types. For both types of data we consider two situations: one where we simulate both the graphical model and the data, and one where the graphical model is given and we only simulate data. For performance evaluation, we demonstrate the use of the SHD, TPR, and the FPRp metrics in a ROC type curve on the pattern graph as described in Section 4.1.2. Each colored curve corresponds to a specific object (id) of an algorithm and the ROC shape comes from different values of a parameter specific to each algorithms, typically the parameter for a score or a test procedure. Since not all methods support the same test procedures or scores, we have chosen the default settings for some of the algorithms. The parameter values are selected so as to indicate how the performance changes. However, some curves are shorter than would be desired due to range limits for the parameter in the programs that were used. For binary data we use the *Bayesian Dirichlet likelihood-equivalence* (BDe) score (Heckerman, Geiger, and Chickering 1995) in the score-based methods, while varying the equivalent sample size parameter (Silander, Kontkanen, and Myllymaki 2012). The benchmarking
| Algorithm         | Space   | Language | Package   | Type   |
|-------------------|---------|----------|-----------|--------|
| Iterative search  | DAG     | R        | BiDAG     | H      |
| Order MCMC        | DAG     | R        | BiDAG     | S      |
| GS                | DAG     | R        | bnlearn   | C      |
| MMHC              | DAG     | R        | bnlearn   | H      |
| HC                | DAG     | R        | bnlearn   | S      |
| Inter-IAMB        | CPDAG   | R        | bnlearn   | C      |
| Tabu              | DAG     | R        | GOBNILP   | S      |
| GOBNILP           | DAG     | C        | GOBNILP   | S      |
| No tears          | DAG     | Python   | notears   | S      |
| PC                | CPDAG   | R        | pcalg     | C      |
| ASOBS             | DAG     | R/Java   | r.blip    | S      |
| FCI               | DAG     | Java     | TETRAD    | C      |
| GFCI              | DAG     | Java     | TETRAD    | H      |
| FGES              | CPDAG   | Java     | TETRAD    | S      |
| RFCI              | CPDAG   | Java     | TETRAD    | C      |

Table 1: Structure learning algorithms for Bayesian networks along with the space on which the algorithm operates (second column), the language the algorithms is implemented in (third column) the package where it is implemented (fourth column), and the type of algorithm (C)onstraint-based, (S)core-based or (H)ybrid (last column).

The exercise considers the following score-based algorithms: HC (hc-bde), TABU (tabu-bde), ASOBS (asobs-bdeu), GOBNILP (gobnilp-bde), iterative search with maximum a posteriori (MAP) estimation (itsearch_map-bde), iterative search with Monte Carlo sampling estimation (itsearch_sample-bde), and order MCMC with the end space of itsearch_sample-bde as start space (omcmc_itsample-bde). Furthermore, we include a number of constraint-based methods: MMHC (mmhc-bde-mi), Inter-IAMB (interiamb-mi), and GS (gs-mi) with the mutual information test; the PC algorithm with the $G^2$-test (pc-binCItest); and the FCI (fci-chi-square), RFCI (rfci-chi-square), and GFCI (gfci-bdeu-chi-square) with the $\chi^2$-test. Since GFCI is a hybrid method it also uses the BDe score. The varying parameter to draw ROC type curves for all constraint based methods is the significance level for the independence tests.

For the continuous datasets, we use two different scores in the score-based methods. The benchmarking exercise in this case includes Tabu (tabu-bge), GOBNILP (gobnilp-bge), HC (hc-bge), iterative search with MAP estimation (itsearch_map-bge), and order MCMC using the end space of itsearch_sample-bge as start space (omcmc_itsample-bge) with the Bayesian Gaussian likelihood-equivalence (BGe) score (Geiger and Heckerman 1994; Geiger, Heckerman et al. 2002; Kuipers, Moffa, Heckerman et al. 2014), and varying levels of equivalent sample size. Furthermore we include the FGES (fges-sem-bic) and GFCI (gfci-sem-bic-fisher-z) with the structural equation model Bayesian information criterion (SEM-BIC) score (Glymour and Scheines 1986), and varying levels of the penalty discount. For constraint-based methods we include the PC algorithm (pc-gaussCItest), GFCI (gfci-sem-bic-fisher-z), FCI (fci-fisher-z), GFCI (gfci-fisher-z), RFCI (rfci-fisher-z), MMHC (mmhc-bge-zf), and Inter-IAMB (interiamb-zf) with the Fisher’s $z$-transformation test of partial correlations (Edwards 2012) with varying significance levels.
5.1. Binary valued Bayesian network with random parameters and random DAG

In this example we study a binary valued Bayesian network, where both the graph $G$ and the parameters $\Theta$ are regarded as random variables. More specifically, we consider 50 models $\{(G_i, \Theta_i)\}_{i=1}^{50}$, where each $G_i$ is sampled according to the Erdős-Rényi random DAG model using the `pcalg_randdag` module (Section 4.2.1), where the number of nodes is $p = 80$, the average number of neighbours (parents) per node is 4 (2) and the maximal number of parents per node is 5, see Listing 2. The parameters $\Theta_i$ are sampled using the `bin_bn` module (Section 4.2.2) and restricting the conditional probabilities within the range $[0.1,0.9]$, see Listing 3. From each model, we draw two datasets $Y_i^{320}$, and $Y_i^{640}$ of sizes $n = 320$ and $n = 640$ using the `iid` module, see Listing 4. The overall data setup is shown in Listing 5.

Figure 1a shows the ROC curves for the algorithms considered for the discrete data as described above, and Listing 6 displays the evaluation section corresponding to this example. We note that for the smaller number of observations ($n = 320$), the results are more spread out in terms of FPRp than for the larger data sets ($n = 640$). For the smaller data sets ($n = 320$), the algorithms standing out in terms of low SHD in combination with low best median TPR ($>0.5$), are FGES (`fges-bdeu`), GFCI (`gfci-bdeu-chi-square`) and iterative order MCMC (`omcmc_itsample-bde`). For the case with larger sample size the same algorithms still achieve higher performance, together with iterative search (`itsearch_sample-bde`).

5.2. Linear Gaussian SEM with random weights and random DAG
In this example we study a continuous random Bayesian network. More specifically, we consider 50 models \( \{(G_i, \Theta_i)\}_{i=1}^{50} \), where each graph \( G_i \) has \( p = 80 \) nodes and is sampled using the \texttt{pcalg\_randdag} module (Section 4.2.1) as described in Section 5.1. The parameters \( \Theta_i \) are sampled from the random linear Gaussian SEM using the \texttt{sem\_params} module (Section 4.2.2) with \( a = 0.25, b = 1, \mu = 0 \) and \( \sigma = 1 \). We draw two standardised datasets \( Y_i^{320} \) and \( Y_i^{640} \), of sizes \( n = 320 \) and \( n = 640 \) (as for the discrete data scenario) from each of the models using the \texttt{iid} module.

Figure 1b shows ROC results for all the algorithms considered for continuous data as described above. For both sample sizes, the constraint based methods FCI (\texttt{fci\_sem\_bic\_fisher\_z}), RFCI (\texttt{rfci\_fisher\_z}), PC (\texttt{pc\_gaussCITest}), and the score based GOBNILP (\texttt{gobnilp\_bge}) have comparable and lower best median TPR (< 0.7) than the remaining algorithms. In terms of achieving high TPR (> 0.7) iterative order MCMC (\texttt{omcmc\_itsample\_bge}) and iterative search MCMC (\texttt{itsearch\_bge}) stand out with near perfect performance, i.e. SHD \( \approx 0 \). Among the other algorithms GFCI performs next best for both sample sizes with TPR \( \approx 0.85 \) and FPRp \( \approx 0.08 \), followed by HC (\texttt{hc\_bge}), MMHC (\texttt{mmhc\_bge\_zf}), FGES (\texttt{fges\_sem\_bic}), and GS (\texttt{gs\_zf}). All algorithms completed except for GOBNILP (\texttt{gobnilp\_bge}), which had to be terminated by hand, even with maximal number of parents set to 2.

5.3. Bayesian network with random parameters and fixed DAG

In this example we study a random binary Bayesian network where the graph \( G \) is fixed and the associated parameters \( \Theta \) are regarded as random. More specifically, we consider 50 models \( \{(G, \Theta_i)\}_{i=1}^{50} \), where the graph structure \( G \) is that of the well known Bayesian network \textit{HEPAR II} (\texttt{hepar2.csv}), first introduced in Onisko (2003). This graph has 70 nodes and
Benchpress: a scalable and platform-independent workflow for structure learning

```json
{
  "roc":
  {
    "ids": ["asobs-bdeu", "gobnilp-bde", "tabu-bde", "mmhc-bde-mi", "hc-bde", "gs-mi", "interiamb-mi", "fci-chi-square", "rfci-chi-square", "gfci-bdeu-chi-square", "fges-bdeu", "itsearch_sample-bde", "pc-binC1test", "omcmc_itsample-bde"],
    "filename_prefix": "examples/section_6.1_",
    "point": true,
    "errorbar": true,
    "path": true,
    "text": false
  },
  "adjmat_true_plots": false,
  "graph_true_plots": false,
  "adjmat_plots": [],
  "graph_plots": [],
  "mcmc_heatmaps": [],
  "mcmc_traj_plots": [],
  "mcmc_autocorr": []
}

Listing 6: The evaluation section for the example in Section 5.1.
```
123 edges and has become one of the standard benchmarks for evaluating the performance of structure learning algorithms. The maximum number of parents per node is 6 and we sample the parameters $\Theta_i$ using the bin_bn module (Section 4.2.2), in the same manner as described in Section 5.1. From each model we draw, as before, two datasets $Y_{i}^{320}$ and $Y_{i}^{640}$ of sample sizes $n = 320$ and $n = 640$, respectively, using the iid module.

Figure 1c shows the ROC curves for this scenario. The algorithms appear to be divided in two groups with respect to their performance in terms of TPR. Constraint based methods including PC (pc-binCIttest), FCI (fci-chi-square), RFCI (rfci-chi-square), inter-IAMB (interiamb-mi), MMHC (mmhc-bde-mi), and GS (gs-mi) appear to cluster in the lower scoring region (TPR < 0.5), with GS achieving the lowest FPRp. Score based methods on the other seem to concentrate in the higher scoring region (TPR > 0.5). For the smaller dataset ($n = 320$), the group of better performing algorithms includes FGES (fges-bdeu), iterative order MCMC (omcmc_itsample-bde), iterative search MCMC (itsearch_sample-bde), GFCI (gfci-bdeu-chi-square), all with FPRp $\approx 0.15$. For the larger data set ($n = 640$) the higher performing group in terms of FPRp $\approx 0.11$ includes again iterative order MCMC (omcmc_itsample-bde) and iterative search MCMC (itsearch_sample-bde), this time joined by GOBNILP (gobnilp-bde).

5.4. Linear Gaussian SEM with random weights and fixed DAG

In this example we draw again 50 models $\{(G, \Theta_i)\}_{i=1}^{50}$, where $G$ corresponds to the HEPAR II network (hepar2.csv), and $\Theta_i$ are the parameters of a linear Gaussian SEM sampled using the sem_params module (Section 4.2.2) with the same settings as in Section 5.2. From each of the models $(G, \Theta_i)$, we draw once more independent data sets $Y_{i}^{320}$ and $Y_{i}^{640}$ of sizes $n = 320$ and $n = 640$ (iid).

The ROC results of Figure 1d highlight that order MCMC (omcmc_itsample-bge), iterative search MCMC (itsearch_sample-bge) visibly separate themselves from the rest in terms of SHD (combining both low FPRp and high TPR) for both sample sizes. In terms of best median TPR (> 0.8) only HC (hc-bge) and Tabu (tabu-bge) display similar performances, while performing considerably worse with respect to median FPRp ($\approx 0.1$ vs $\approx 0.17$). There is in general not a big difference in performance of the algorithms between the two sample sizes. All algorithm completed except for GOBNILP (gobnilp-bge), which again had to be terminated by hand, even with maximal number of parents set to 2.

6. Using the software

Benchpress is tested on Linux systems using snakemake $\geq 5.2$, where singularity $\geq 3.5$ (Kurtzer et al. 2017) is installed and used as container orchestration manager. While benchpress in principal works on any Unix-like system (OSX/Linux), Singularity is at the time of writing only fully supported by Linux systems. However, Snakemake indicates that also other container orchestration managers such as podman will be supported in the future.

6.1. Installation and usage

With the requirements above installed, the easiest way to get started with benchpress is to download or clone the git repository from https://github.com/felixleopoldo/benchpress
and use the example config file config/ex.json. **Benchpress** can then be run through standard **snakemake** commands and we refer to (Köster and Rahmann 2012) for further details. The steps are summarised as follows:

```
$ git clone https://github.com/felixleopoldo/benchpress.git
$ cd benchpress
$ snakemake --cores all --use-singularity --configfile config/ex.json
```

If **snakemake** is installed with **conda** (Anaconda 2016) in an environment named e.g. **snakemake**, then it should first be activated by the following command:

```
$ conda activate snakemake
```

### 6.2. Reproducibility

Following the pattern suggested by **snakemake**, the version of **benchpress** that was used to generate this study was saved as a tarball (**benchpress_paper.tar.gz**) using the command

```
$ snakemake --archive benchpress_paper.tar.gz
```

It was then published at [http://zenodo.org](http://zenodo.org) where it was assigned its own digital object identifier (DOI) (Rios, Moffa, and Kuipers 2021). To reproduce the results of Section 5 simply download the tarball from [https://doi.org/10.5281/zenodo.5226238](https://doi.org/10.5281/zenodo.5226238) and run the simulations by typing e.g.

```
$ mkdir benchpress_paper
$ tar -xf benchpress_paper.tar.gz -C benchpress_paper
$ cd benchpress_paper
$ snakemake --cores all --use-singularity --configfile config/sec6.1.json
```

However, this study is rather large (∼ 8000 jobs) so laptop users are recommended to use a shorter **seed_range** and only a subset of the algorithms.

### 7. Conclusions

**Benchpress** provides a novel **snakemake** workflow for scalable and re-producible execution and benchmarking of structure learning algorithms. **Snakemake**’s support for running containerized software together with the simple data and graph representation enables **benchpress** to compare algorithms implemented in different programming languages. In its current version **benchpress** already implements 19 of the state-of-the-art learning algorithms, as well as a number of modules for generating data models and evaluating performance. As **benchpress** is built in a completely modular form **snakemake** allows for seamless scaling of computations over multiple cores, grids or servers without any extra additional effort. In addition, it is straightforward to integrate new modules into the software. Even though the **benchpress** project so far has a focus on structure learning for graphical models, we also see the potential in extending **benchpress** to evaluate other statistical models.
Acknowledgments

We would like to thank all the developers and researchers that have made their software available in the public domain.

A.

A.1. Data formats

Throughout this section we assume a four dimensional graphical model where the nodes are labeled as $a, b, c$ and $d$.

Data set

Observations should be stored as row vectors in a matrix, where the columns are separated by a comma. The first row is special and should contain the labels of the variables. If the data is categorical, the second row should contain the cardinality (number of levels) of each variable.

Below is a formatting example of two samples of a categorical distribution where the cardinalities are 2,3,2, and 2.

\[
\begin{array}{cccc}
\text{a, b, c, d} \\
2,3,2,2 \\
1,2,0,1 \\
0,1,1,1 \\
\end{array}
\]

An example showing of two samples from continuous distribution is shown below.

\[
\begin{array}{cccc}
\text{a, b, c, d} \\
0.2,2.3,5.3,0.5 \\
3.2,1.5,2.5,1.2 \\
\end{array}
\]

Adjacency matrix

A graph $G$ in benchpress is represented as adjacency matrix $M$, where $M_{ij} = 1$ if $(i, j) \in E$ and $M_{ij} = 0$ if $(i, j) \notin E$. An undirected graph is represented by a symmetric matrix.

Below is an example undirected graph $G = (V, E)$, where $E = \{(a, b), (a, c), (c, d)\}$ are interpreted as un-ordered pairs (un-directed edges).

\[
\begin{array}{cccc}
\text{a, b, c, d} \\
0,1,1,0 \\
1,0,0,0 \\
1,0,0,1 \\
0,0,1,0 \\
\end{array}
\]

If $G$ is directed the adjacency matrix is asymmetric as below.
Benchpress: a scalable and platform-independent workflow for structure learning

MCMC trajectory

When the output of the algorithm is a Markov chain of graphs, we store the output in a compact form by tracking only the changes when moves are accepted, along with the corresponding time index and the score of the resulting graph after acceptance (not the score difference).

Additionally, in the first two rows the labels of the variables, which should be read from the data matrix, are recorded. Specifically, the first row (index -2) contains edges from the first variable to each of the rest in the added column, where a dash (−) symbolises an undirected edge, and a right arrow (→) a directed edge. The score column is set to 0 and removed is set to []. The second row (index -1) has the same edges in the removed column, while the score column is set to 0 and added is set to []. The third row (index 0) contains all the vertices in the starting graph along with its score in the score column and [] in the removed column.

Below is an example of a trajectory of undirected graphs \( G_0, G_1, \ldots, G_{89} \), where \( E_i = \{(b, c), (a, d)\} \) for \( i = 0, \ldots, 33 \), \( E_i = \{(a, d)\} \) for \( i = 34, \ldots, 88 \) and \( E_i = \{(c, d), (a, d)\} \) for \( i = 89 \).

| index, score, added, removed |
|-------------------------------|
| -2, 0.0, [a-b; a-c; a-d], []  |
| -1, 0.0, [], [a-b; a-c; a-d]  |
| 0, -2325.52, [b-c; a-d], []  |
| 34, -2311.94, [], [b-c]      |
| 89, -2310.81, [c-d], []      |

References

Anaconda (2016). “Anaconda Software Distribution.”

Atay-Kayis A, Massam H (2005). “A Monte Carlo method for computing the marginal likelihood in nondecomposable Gaussian graphical models.” Biometrika, 92(2), 317–335.

Boettiger C, Edelbuettel D (2017). “An introduction to rocker: Docker containers for R.” arXiv preprint arXiv:1710.03675.

Carvalho CM (2006). Structure and sparsity in high-dimensional multivariate analysis, volume 68.

Carvalho CM, Massam H, West M (2007). “Simulation of hyper-inverse Wishart distributions in graphical models.” Biometrika, 94(3), 647–659.
Chickering DM (1995). “A transformational characterization of equivalent Bayesian network structures.” *arXiv preprint arXiv:1302.4938.*

Chickering DM (2002). “Optimal structure identification with greedy search.” *Journal of machine learning research, 3*(Nov), 507–554.

Chickering DM, Heckerman D, Meek C (2004). “Large-sample learning of Bayesian networks is NP-hard.” *Journal of Machine Learning Research, 5.*

Colombo D, Maathuis MH, Kalisch M, Richardson TS (2012). “Learning high-dimensional directed acyclic graphs with latent and selection variables.” *The Annals of Statistics*, pp. 294–321.

Constantinou AC (2020). “Learning Bayesian networks that enable full propagation of evidence.” *arXiv preprint arXiv:2004.04571.*

Constantinou AC, Liu Y, Chobtham K, Guo Z, Kitson NK (2020). “The Bayesys data and Bayesian network repository.” *Queen Mary University of London*, pp. 2–2.

Constantinou AC, Liu Y, Chobtham K, Guo Z, Kitson NK (2021). “Large-scale empirical validation of Bayesian network structure learning algorithms with noisy data.” *International Journal of Approximate Reasoning, 131*, 151–188.

Cowell RG, Dawid P, Lauritzen SL, Spiegelhalter DJ (2003). *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks.* Information Science and Statistics. Springer New York. ISBN 9780387987675.

Cussens J (2012). “Bayesian network learning with cutting planes.” *arXiv preprint arXiv:1202.3713.*

Cussens J (2020). “GOBNILP: Learning Bayesian network structure with integer programming.” In *International Conference on Probabilistic Graphical Models*, pp. 605–608. PMLR.

Dawid AP, Lauritzen SL (1993). “Hyper Markov laws in the statistical analysis of decomposable graphical models.” *The Annals of Statistics, 21*(3), 1272–1317.

Diestel R (2005). *Graph theory (Graduate texts in mathematics)*, volume 173. Springer Heidelberg.

Duarte E, Solus L (2021). “A new characterization of discrete decomposable models.” *arXiv preprint arXiv:2105.05907.*

Edwards D (2012). *Introduction to graphical modelling.* Springer Science & Business Media.

Elmasri M (2017). “On decomposable random graphs.” *ArXiv e-prints.*

Elwert F (2013). “Graphical causal models.” In *Handbook of causal analysis for social research*, pp. 245–273. Springer.

Friedman J, Hastie T, Tibshirani R (2008). “Sparse inverse covariance estimation with the graphical lasso.” *Biostatistics, 9*(3), 432–441. ISSN 1468-4357.
Friedman N (2004). “Inferring cellular networks using probabilistic graphical models.” Science, 303(5659), 799–805.

Friedman N, Goldszmidt M, Heckerman D, Russell S (1997). “Where is the impact of Bayesian networks in learning.” In International Joint Conference on Artificial Intelligence. Citeseer.

Friedman N, Koller D (2003). “Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks.” Machine Learning, 50(1), 95–125. ISSN 1573-0565.

Friedman N, Linial M, Nachman I, Pe’er D (2000). “Using Bayesian networks to analyze expression data.” Journal of computational biology, 7(3-4), 601–620.

Geiger D, Heckerman D (1994). “Learning Gaussian networks.” In Proceedings of the Tenth international conference on Uncertainty in artificial intelligence, pp. 235–243. Morgan Kaufmann Publishers Inc.

Geiger D, Heckerman D, et al. (2002). “Parameter priors for directed acyclic graphical models and the characterization of several probability distributions.” The Annals of Statistics, 30(5), 1412–1440.

Giudici P, Castelo R (2003). “Improving Markov chain Monte Carlo model search for data mining.” Machine learning, 50(1), 127–158.

Giudici P, Green PJ (1999). “Decomposable graphical Gaussian model determination.” Biometrika, 86(4), 785–801.

Glymour C, Scheines R (1986). “Causal modeling with the TETRAD program.” Synthese, 68(1), 37–63.

Green PJ, Thomas A (2013). “Sampling decomposable graphs using a Markov chain on junction trees.” Biometrika, 100(1), 91–110.

Heckerman D, Geiger D, Chickering DM (1995). “Learning Bayesian networks: The combination of knowledge and statistical data.” Machine learning, 20(3), 197–243.

Jennings D, Corcoran JN (2018). “A birth and death process for Bayesian network structure inference.” Probability in the Engineering and Informational Sciences, 32(4), 615–625.

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.

Koller D, Friedman N (2009). Probabilistic Graphical Models - Principles and Techniques. MIT Press. ISBN 978-0-262-01319-2.

Koski T, Noble J (2012). “A review of Bayesian networks and structure learning.” Mathematica Applicanda, 40, 51–103.

Köster J, Rahmann S (2012). “Snakemake—a scalable bioinformatics workflow engine.” Bioinformatics, 28(19), 2520–2522. ISSN 1367-4803.
Kuipers J, Moffa G (2017). “Partition MCMC for inference on acyclic digraphs.” *Journal of the American Statistical Association*, **112**(517), 282–299.

Kuipers J, Moffa G, Heckerman D, *et al.* (2014). “Addendum on the scoring of Gaussian directed acyclic graphical models.” *Annals of Statistics*, **42**(4), 1689–1691.

Kuipers J, Moffa G, Kuipers E, Freeman D, Bebbington P (2019). “Links between psychotic and neurotic symptoms in the general population: an analysis of longitudinal British national survey data using directed acyclic graphs.” *Psychological Medicine*, **49**(3), 388–395.

Kuipers J, Suter P, Moffa G (2018a). “Efficient structure learning and sampling of Bayesian networks.” *arXiv preprint arXiv:1803.07859*.

Kuipers J, Thurnherr T, Moffa G, Suter P, Behr J, Goosen R, Christofori G, Beerenwinkel N (2018b). “Mutational interactions define novel cancer subgroups.” *Nature communications*, **9**(1), 1–10.

Kurtzer GM, Sochat V, Bauer MW (2017). “Singularity: Scientific containers for mobility of compute.” *PloS one*, **12**(5), e0177459.

Lamprecht AL, Garcia L, Kuzak M, Martinez C, Arcila R, Martin Del Pico E, Domínguez Del Ángel V, van de Sandt S, Ison J, Martinez PA, McQuilton P, Valencia A, Harrow J, Psomopoulos F, Gelpi JL, Chue Hong N, Goble C, Capella-Gutierrez S (2020). “Towards FAIR principles for research software.” *3*, 37–59.

Lauritzen SL (1996). *Graphical Models*. Oxford University Press. ISBN 0-19-852219-3.

Lenkoski A (2013). “A direct sampler for G-Wishart variates.” *Stat*, **2**(1), 119–128.

Madigan D, York J, Allard D (1995). “Bayesian Graphical Models for Discrete Data.” *International Statistical Review / Revue Internationale de Statistique*, **63**(2), 215–232.

Margaritis D (2003). “Learning Bayesian network model structure from data.” *Technical report*, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science.

Meek C (1997). *Graphical Models: Selecting causal and statistical models*. Ph.D. thesis, PhD thesis, Carnegie Mellon University.

Meinshausen N (2008). “A note on the Lasso for Gaussian graphical model selection.” *Statistics and Probability Letters*, **78**(7), 880 – 884. ISSN 0167-7152.

Merkel D (2014). “Docker: lightweight linux containers for consistent development and deployment.” *Linux journal*, **2014**(239), 2.

Moffa G, Catone G, Kuipers J, Kuipers E, Freeman D, Marwaha S, Lennox BR, Broome MR, Bebbington P (2017). “Using Directed Acyclic Graphs in Epidemiological Research in Psychosis: An Analysis of the Role of Bullying in Psychosis.” *Schizophrenia Bulletin*, **43**(6), 1273–1279. ISSN 0586-7614.

Mohammadi R, Wit EC (2019). “BDgraph: An R Package for Bayesian Structure Learning in Graphical Models.” *Journal of Statistical Software, Articles*, **89**(3), 1–30. ISSN 1548-7660.
Munafò MR, Nosek BA, Bishop DVM, Button KS, Chambers CD, Percie du Sert N, Simonsohn U, Wagenmakers EJ, Ware JJ, Ioannidis JPA (2017). “A manifesto for reproducible science.” *Nature Human Behaviour*, 1(1), 0021.

Niist D, Sochat V, Marwick B, Eglen SJ, Head T, Hirst T, Evans BD (2020). “Ten simple rules for writing Dockerfiles for reproducible data science.” *PLOS Computational Biology*, 16(11), 1–24.

Ogarrio JM, Spirtes P, Ramsey J (2016). “A hybrid causal search algorithm for latent variable models.” In *Conference on Probabilistic Graphical Models*, pp. 368–379.

Olsson J, Pavlenko T, Rios FL (2018). “Sequential sampling of junction trees for decomposable graphs.” *ArXiv e-prints*.

Olsson J, Pavlenko T, Rios FL (2019). “Bayesian learning of weakly structural Markov graph laws using sequential Monte Carlo methods.” *Electron. J. Statist.*, 13(2), 2865–2897.

Onisko A (2003). *Probabilistic causal models in medicine: Application to diagnosis in liver disorders*. Institute of Biocybernetics and Biomedical Engineering. Ph.D. thesis, PhD dissertation. Polish Academy of Science.

Pearl J (1995). “Causal diagrams for empirical research.” *Biometrika*, 82(4), 669–688.

Pearl J (1997). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Representation and Reasoning Series. Morgan Kaufmann. ISBN 9781558604797.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. (2011). “Scikit-learn: Machine learning in Python.” *the Journal of machine Learning research*, 12, 2825–2830.

Pezoa F, Reutter JL, Suarez F, Ugarte M, Vrgoč D (2016). “Foundations of JSON schema.” In *Proceedings of the 25th International Conference on World Wide Web*, pp. 263–273. International World Wide Web Conferences Steering Committee.

Ramsey JD, Malinsky D, Bui KV (2020). “algcomparison: Comparing the Performance of Graphical Structure Learning Algorithms with TETRAD.” *Journal of Machine Learning Research*, 21(238), 1–6.

Rantanen K, Hyttinen A, Järvisalo M, et al. (2017). “Learning chordal Markov networks via branch and bound.” *Advances in Neural Information Processing Systems 30 (NIPS 2017).*

Reisach AG, Seiler C, Weichwald S (2021). “Beware of the Simulated DAG! Varsortability in Additive Noise Models.” *arXiv preprint arXiv:2102.13647*.

Rios FL, Moffa G, Kuipers J (2021). “Benchpress paper version.” doi: 10.5281/zenodo.5226238. URL https://doi.org/10.5281/zenodo.5226238.

Russell S, Norvig P (2002). “Artificial intelligence: a modern approach.”

Scanagatta M, Corani G, De Campos CP, Zaffalon M (2018). “Approximate structure learning for large Bayesian networks.” *Machine Learning*, 107(8-10), 1209–1227.
Scanagatta M, de Campos CP, Corani G, Zaffalon M (2015). “Learning Bayesian networks with thousands of variables.” In *Advances in neural information processing systems*, pp. 1864–1872.

Scutari M (2010). “Learning Bayesian Networks with the bnlearn R Package.” *Journal of Statistical Software, Articles*, 35(3), 1–22. ISSN 1548-7660.

Scutari M, Graafland CE, Gutiérrez JM (2019a). “Who learns better Bayesian network structures: Accuracy and speed of structure learning algorithms.” *International Journal of Approximate Reasoning*, 115, 235–253.

Scutari M, Vitolo C, Tucker A (2019b). “Learning Bayesian networks from big data with greedy search: computational complexity and efficient implementation.” *Statistics and Computing*, 29(5), 1095–1108.

Silander T, Kontkanen P, Myllymaki P (2012). “On sensitivity of the MAP Bayesian network structure to the equivalent sample size parameter.” *arXiv preprint arXiv:1206.5293*.

Spirtes P, Glymour CN (1991). “An algorithm for fast recovery of sparse causal graphs.” *Social science computer review*, 9(1), 62–72.

Spirtes P, Glymour CN, Scheines R, Heckerman D (2000). *Causation, prediction, and search*. MIT press.

Studený M, Cussens J (2017). “Towards using the chordal graph polytope in learning decomposable models.” *International Journal of Approximate Reasoning*, 88, 259–281.

Suter P, Kuipers J, Moffa G, Beerenwinkel N (2021). “Bayesian structure learning and sampling of Bayesian networks with the R package BiDAG.” *arXiv preprint arXiv:2105.00488*.

Teyssier M, Koller D (2012). “Ordering-based search: A simple and effective algorithm for learning Bayesian networks.” *arXiv preprint arXiv:1207.1429*.

Tsamardinos I, Aliferis CF, Statnikov AR, Statnikov E (2003). “Algorithms for large scale Markov blanket discovery.” In *FLAIRS conference*, volume 2, pp. 376–380.

Tsamardinos I, Brown LE, Aliferis CF (2006). “The max-min hill-climbing Bayesian network structure learning algorithm.” *Machine learning*, 65(1), 31–78.

Verma T, Pearl J (1991). *Equivalence and synthesis of causal models*. UCLA, Computer Science Department.

Waskom ML (2021). “seaborn: statistical data visualization.” *Journal of Open Source Software*, 6(60), 3021.

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). “Welcome to the tidyverse.” *Journal of Open Source Software*, 4(43), 1686.

Witten DM, Friedman JH, Simon N (2011). “New Insights and Faster Computations for the Graphical Lasso.” *Journal of Computational and Graphical Statistics*, 20(4), 892–900.
Wormald NC (1985). “Counting labelled chordal graphs.” *Graphs and Combinatorics*, 1(1), 193–200. ISSN 0911-0119.

Zheng X, Aragam B, Ravikumar PK, Xing EP (2018). “DAGs with NO TEARS: Continuous optimization for structure learning.” In *Advances in Neural Information Processing Systems*, pp. 9472–9483.

**Affiliation:**

Felix L. Rios  
Department of Mathematics and Computer Science  
University of Basel  
Spiegelgasse 1, 4051 Basel, Switzerland  
E-mail: felixleopoldo.rios@unibas.ch  
URL: [http://github.com/felixleopoldo](http://github.com/felixleopoldo)

Giusi Moffa  
Department of Mathematics and Computer Science  
University of Basel  
Spiegelgasse 1, 4051 Basel, Switzerland  
E-mail: giusi.moffa@unibas.ch

Jack Kuipers  
Department of Biosystems Science and Engineering  
ETH Zürich  
Mattenstrasse 26, 4058 Basel, Switzerland  
E-mail: jack.kuipers@bsse.ethz.ch
Figure 1: ROC curves showing the FPRp and TPR evaluated on 50 datasets from 4 different Bayesian network models. Each color corresponds to a structure learning algorithm with object id shown to the right.
This figure "jsslogo.jpg" is available in "jpg" format from:

http://arxiv.org/ps/2107.03863v2