Scaling Up Bayesian DAG Sampling

Daniele Nikzad[1] **Alexander Zhilkin**[1] **Juha Harviainen**[1] **Jack Kuipers**[2] **Giusi Moffa**[3] **Mikko Koivisto**[1]

[1]University of Helsinki
[2]ETH Zurich
[3]University of Basel

Abstract

Bayesian inference of Bayesian network structures is often performed by sampling directed acyclic graphs along an appropriately constructed Markov chain. We present two techniques to improve sampling. First, we give an efficient implementation of basic moves, which add, delete, or reverse a single arc. Second, we expedite summing over parent sets, an expensive task required for more sophisticated moves: we devise a preprocessing method to prune possible parent sets so as to approximately preserve the sums. Our empirical results indicate that our techniques can yield substantial efficiency gains compared to previous methods.

1 INTRODUCTION

Directed acyclic graphs (DAGs) provide a concise way to express conditional independence relations among random variables, decomposing a multivariate distribution into a product of univariate conditional distributions (Pearl, 1988). Such models, known as Bayesian networks or belief networks (BNs), can also be naturally attached with causal semantics, with an arc from one node to another in the graph corresponding to a (possible) direct cause–effect relation between the associated variables (Pearl, 2009).

Numerous methods have been proposed to learn DAGs from data, usually modelling the data points as independent draws from an unknown BN, but otherwise following various learning paradigms (Kitson et al., 2023; Koller and Friedman, 2009, Ch. 18; Rios et al., 2025). The natural approach to account for the uncertainty is the Bayesian one, which formalizes learning as the transformation of a prior distribution over DAGs into a posterior (Buntine, 1991; Cooper and Herskovits, 1992; Heckerman et al., 1995). Within this paradigm, different methods are mainly ranked by their *computational efficiency*, that is, how fast they can (reliably) produce a good approximation of the posterior, exact algorithms being feasible for networks on only about twenty or fewer variables (Harviainen and Koivisto, 2024; Koivisto and Sood, 2004; Pensar et al., 2020; Talvitie et al., 2020; Tian and He, 2009).

Approximating a DAG posterior also serves as a concrete challenge for general-purpose methodology for probabilistic inference. The most prominent approaches taken so far are Markov chain Monte Carlo (MCMC; see, e.g., Friedman and Koller, 2003; Giudici and Castelo, 2003; Grzegorczyk and Husmeier, 2008; Kuipers et al., 2022; Madigan and York, 1995; Su and Borsuk, 2016), variational inference (Annadani et al., 2023; Cundy et al., 2021; Lorch et al., 2021), and generative flows (Deleu et al., 2022, 2023); while MCMC is based on sampling that converges to the posterior, the latter two aim at learning a function that approximates the posterior. Ideas that prove useful for dealing with the hard-to-handle space of DAGs, therefore, have the potential to be valuable also more generally.

Here, we advance *structure MCMC*, one of the most promising approaches to reliable Bayesian sampling of DAGs. Due to certain computational bottlenecks, however, the existing implementations do not scale up to larger DAGs and larger datasets. Using our proposed improvements, we demonstrate the competitiveness of structure MCMC through experimental comparisons with alternative approaches in Section 5. Our algorithmic ideas not only improve DAG sampling, but may also find applications in other related contexts.

We will begin our advancements with a basic Markov chain that moves by adding, removing, or reversing a single arc. Giudici and Castelo (2003) gave an efficient way to simulate such a chain, which we refer to as the *GC* algorithm. It maintains the ancestor relation of the DAG to enable quick acyclicity checks. In Section 3, we significantly expedite *GC*, by *one to three orders of magnitude* in our experiments. Our key idea is to use an appropriate upper bound for the (typically small) acceptance probability of the proposed move, so that the time staying in the current state can be

Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence (UAI 2026), PMLR 337:4949–4971.

simulated simply by a geometric random variable.

In Section 4, we consider improved Markov chains that, besides basic moves, also resample the entire parent set for certain nodes (Goudie and Mukherjee, 2016; Grzegorczyk and Husmeier, 2008; Su and Borsuk, 2016). While such moves can make bigger changes to the DAG, they also require computing sums of local scores over large numbers of possible parent sets. The same challenge is encountered with MCMC methods that primarily sample node orderings instead of DAGs (Friedman and Koller, 2003; Kuipers and Moffa, 2017; Kuipers et al., 2022; Niinimäki and Koivisto, 2013; Viinikka and Koivisto, 2020; Viinikka et al., 2020). Previous works have addressed this challenge by limiting the number of parents, by only allowing parents from a few preselected candidates, and by accumulating the sums in decreasing order of scores until a reasonable approximation is achieved (Friedman and Koller, 2003; Kuipers et al., 2022; Niinimäki and Koivisto, 2013; Viinikka et al., 2020). Here, we add a new tool: a pruning method that discards parent sets whose local scores can be proven not to contribute significantly to any relevant score sum. Our experiments show significant savings, with improvements again reaching *one to three orders of magnitude*.

In Section 5, we integrate the new techniques into an enhanced implementation of structure MCMC we dub *Gibby* (available at `https://github.com/sums-of-products/gibby`) and study its performance against other available Bayesian DAG samplers. We stress that our DAG sampler is agnostic to the type of data and modeling assumptions: the sampling phase depends solely on unnormalized log-probabilities, which can be computed under various statistical models, shaping the posterior accordingly. Thus, our primary interest is not in *modeling and learning accuracy*: neither in finding a single DAG that "best fits the data," nor in learning arcs or other features of the data-generating graph (even when one exists). Our interest is in *computational complexity*: how reliably and fast the samplers can produce a good approximation of the well-defined DAG posterior. Throughout the paper, we illustrate our findings using datasets generated from selected benchmark BNs (Scutari, 2022).

2 PRELIMINARIES

We begin by recalling the basics of Bayesian learning of BNs, and then review the previous development of structure MCMC, on which we build in later sections.

2.1 BAYESIAN LEARNING OF DAGS

Consider a vector $x = (x_1, \ldots, x_n)$ of n random variables. A BN represents a joint probability distribution of x as a pair (G, f), where $G = (N, A)$ is a directed acyclic graph (DAG) on the node set $N = \{1, \ldots, n\}$ and arc set A, and f factorizes into a product of univariate conditional distributions along the graph: $f(x) = \prod_{i \in N} f(x_i | x_{A_i})$. Here $A_i = \{j : ji \in A\}$ is the parent set of i in G, and for any set S we write x_S for the tuple $(x_j : j \in S)$. In what follows, the node set N will be fixed, whereas the arc set A varies. We may thus identify a DAG G with its arc set A.

To learn a BN (G, f) from a data set X of m points $x^{(1)}, \ldots, x^{(m)}$, we model them as independent draws from f. Taking a Bayesian approach, we build a joint model by multiplying the likelihood $P(X|G, f) = \prod_{t=1}^{m} f(x^{(t)})$ by a prior $P(G, f) = P(G)P(f|G)$. The posterior $P(G, f|X)$ is the outcome of learning.

In the experiments, we focus on discrete data and adopt the following forms, unless stated otherwise. For the structure prior, we use $P(G) \propto c^{-|A|}$, with either $c = 1$ or $c = n$; the former is uniform over all DAGs and thus concentrates almost all mass on dense graphs; the latter, we call *sparse*, renders large parent sets unlikely.[1] For the prior on f, we assume each x_i is categorical, parameterize f by full conditional probability tables, and assign the parameters independent Dirichlet priors. Then the marginal likelihood has a closed-form expression, the Bayesian Dirichlet (BD) score (Buntine, 1991; Heckerman et al., 1995), or the *BDeu score* when parameterized by a single constant called the *equivalent sample size*. These choices enable efficient computation of $P(G|X)$ for any given G up to a normalizing constant, as $\pi(G) = \prod_i \pi_i(A_i)$. We call the factors $\pi_i(A_i)$ in this product *local scores*.

In Suppl.E.4, we demonstrate *Gibby* on continuous data using a linear Gaussian model. In this case, we use the Bayesian Gaussian equivalent (BGe) score (Geiger and Heckerman, 1994; Kuipers et al., 2014), which is a closed-form expression for the marginal likelihood.

2.2 SAMPLING ALONG MARKOV CHAINS

The Metropolis–Hasting (MH) algorithm (Hastings, 1970) gives us a way to construct a Markov chain on the state space of DAGs such that its stationary distribution is $P(G|X)$. First, an initial state G_0 is generated arbitrarily. Then, iteratively, a new state G' is in step $t + 1$ drawn from a proposal distribution $Q(G'|G_t)$ and accepted as the next state G_{t+1} with probability $\min\{1, R(G', G_t)\}$, where

$$R(G', G_t) = \frac{\pi(G')Q(G_t|G')}{\pi(G_t)Q(G'|G_t)} \tag{1}$$

[1] For a node with k parents, the term in the sparse prior is $n^{-k} \approx \binom{n-1}{k}^{-1}/k!$. With the term $\binom{n-1}{k}^{-1}$ alone, the prior is roughly uniform over parent set sizes (Eggeling et al., 2019; Friedman and Koller, 2003). The mean of the distribution $p_k \propto 1/k!$ on $k \in \mathbb{N}$ is 1. More generally, by replacing n by n/μ in the sparse prior, the expected indegree per node becomes roughly μ.

is the untruncated acceptance ratio. If the proposal is rejected, G_{t+1} is set to G_t.

While very mild conditions on the proposal distribution Q suffice for guaranteeing convergence to the posterior, a good choice of Q can significantly improve the mixing of the chain and, thereby, expedite convergence. Typically $Q(G'|G_t)$ is chosen to have a restricted support: the chain can move only to some *neighbors* of G_t, as specified by the available move types. We next review move types proposed in the literature, in increasing order of complexity.

The *basic moves* consist of adding, deleting, or reversing a single arc. A naive implementation would explicitly construct the neighborhood of G_t and then draw one neighbor uniformly at random. This is computationally demanding, as typically one could add $\Omega(n^2)$ different arcs. Moreover, to check whether a candidate arc can actually be added, one has to check for acyclicity of the resulting graph: effectively, one has to check whether the head node of the arc is an ancestor of the tail node in the original graph, which can take time $\Omega(\ell)$ in a graph with ℓ arcs.

Giudici and Castelo (2003) address the computational challenge by maintaining the ancestor relation in an $n \times n$ binary matrix. Node pairs are tentatively proposed for addition or reversal—acyclicity is checked afterwards. Testing whether an arc ij can be added or reversed takes, respectively, $O(1)$ or $O(\ell_j)$ time when node j has ℓ_j parents. Only after accepting a move, the matrix is updated in $O(n\ell)$ time.[2]

Grzegorczyk and Husmeier (2008) presented a more involved *edge reversal* (REV) move. Not only an arc ij, selected uniformly at random, is reversed, but also the parent sets of both end nodes i and j are resampled based on the local scores and ensuring acyclicity. The ratio (1) for moving from $G = (N, A)$ to $G' = (N, A')$ becomes

$$R(G', G) = \frac{|A|}{|A'|} \frac{Z'_{\text{tail},ji}}{Z_{\text{tail},ij}} \frac{Z'_{\text{head},ji}}{Z_{\text{head},ij}},$$

where each of the four *Zustandssumme*-terms is the normalizing constant for drawing the respective parent set. More precisely, letting U_i denote the set of *non-descendants* of node i in G, we have

$$Z_{\text{tail},ij} = \sum_{S \subseteq U_i} \pi_i(S), \quad Z_{\text{head},ij} = \sum_{i \in S \subseteq U_j} \pi_j(S),$$

and similar sum formulas for $Z'_{\text{head},ji}$ and $Z'_{\text{tail},ji}$.

Su and Borsuk (2016) took the idea of resampling parent sets further. Their *Markov blanket resampling* (MBR) move first draws a random target node i and a random ordering of its children. Next the arcs to i are removed, as well as the arcs to each child j of i, with the exception of the arcs outgoing from i. Then a new parent set of i, disjoint from the original A_i, is sampled subject to the acyclicity requirement. Last, new parents are sampled for each child one by one, in the fixed ordering, ensuring that i is one parent and the resulting graph is acyclic.

For a more precise description, denote by $G_{(j)}$ the graph obtained from G by removing all arcs to child j of i and to all nodes succeeding j in the ordering. Denote by $U_{(j)}$ the set of non-descendants of j in $G_{(j)}$. Then the ratio (1) can be written as

$$R(G', G) = \frac{Z'_i}{Z_i} \prod_{j \in C_i} \frac{Z'_{(j)}}{Z_{(j)}},$$

where C_i is the set of children of i, and

$$Z_i = \sum_{S \subseteq U_i \setminus A_i} \pi_i(S), \quad Z_{(j)} = \sum_{i \in S \subseteq U_{(j)}} \pi_j(S),$$

and similar sum formulas for Z'_i and $Z'_{(j)}$.

While REV and MBR moves can take longer leaps in the space of DAGs (with reasonable probability), compared to basic moves, they are also computationally more demanding. Even if the number of parents per node is bounded by some constant d, the Z-terms generally involve $\Omega(n^d)$ parent sets.

This is the bottleneck also in yet another similar move proposed by Goudie and Mukherjee (2016). Their block update move resamples the parent sets for a few (say, three) randomly selected nodes. It differs from REV and MBR in that Gibbs sampling is used.

3 FAST BASIC MOVES

This section presents a novel algorithm that, like the one by Giudici and Castelo (2003), performs three basic moves: removing, adding, and reversing an arc. It simulates the same Markov chain, however, leveraging a different architecture and proposal distribution. Both methods rely on rejection sampling: a move may propose a cyclic graph, but such proposals are rejected upon detecting cyclicity. The distinctive feature of our algorithm is that acyclic proposals always get accepted! Thus, we do not waste simulation steps for proposing low-scoring DAGs that would get rejected, like the *GC* algorithm does. We call our algorithm *Gibby*, as it bears some resemblance to Gibbs sampling.

3.1 ENHANCED SIMULATION

Let us begin with the *GC* algorithm to highlight its weaknesses that motivated the development of our new algorithm. The *GC* algorithm proceeds as follows. It first draws a node pair ij according to a fixed probability distribution (q_{ij}), independent of G. Then it proposes a move to G^{ij}, obtained

[2] Giudici and Castelo (2003) claim $O(n)$ update time after addition and $O(n^2)$ after removal; however, the correct bounds for their algorithms appear to be $O(n^2)$ and $O(n\ell)$.

from G either by removing arc ij (if present in G), reversing[3] arc ji (if present in G), or adding arc ij (if neither ij nor ij is in G). Finally, G^{ij} is accepted as the next state of the chain with probability

$$\alpha(G^{ij}, G) = \min\left\{1, \frac{\pi(G^{ij})}{\pi(G)}\right\},$$

where $\pi(G^{ij}) = 0$ if G^{ij} is cyclic; otherwise the next state is G. It is easy to show that if $q_{ij} = q_{ji}$, the chain converges to its stationary distribution π. In our experiments, we used the uniform distribution, $1/q_{ij} = n(n-1)$; with some other choice, one may concentrate proposals, e.g., on node pairs that are more likely adjacent in the DAG posterior.

A major weakness of the above algorithm is that it may frequently propose low-scoring graphs G^{ij}, which get rejected with high probability. Ideally, we would generate G^{ij} with probability $a_{ij} := q_{ij} \cdot \alpha(G^{ij}, G)$. With probability $a := \sum_{ij} a_{ij}$ the chain would move to some G^{ij} and with probability $1 - a$ it would stay at G. This involves visiting and performing acyclicity checks on each possible G^{ij}, which would be computationally demanding.

We give an efficient variant of the above scheme. The key is to consider *tentative acceptance probabilities*

$$\beta(G^{ij}, G) = \min\left\{1, \frac{\pi^*(G^{ij})}{\pi(G)}\right\},$$

where we obtain π^* from π by extending the function to *all digraphs*, i.e., ignoring the acyclicity constraint. In particular, if $G^{ij} = (N, A')$, with A' possibly cyclic, we have

$$\frac{\pi^*(G^{ij})}{\pi(G)} = \frac{\pi_i(A'_i)}{\pi_i(A_i)} \frac{\pi_j(A'_j)}{\pi_j(A_j)}.$$

Note that if G^{ij} is obtained by deleting or adding arc ij from node i to node j, then $A'_i = A_i$ and the respective ratio cancels out, leaving just the ratio $\pi_j(A'_j)/\pi_j(A_j)$.

Now, let

$$b_{ij} := q_{ij} \cdot \beta(G^{ij}, G), \quad b := \sum_{ij} b_{ij}.$$

To simulate the next state after G, we let the chain stay at G with probability $1 - b$. With the remaining probability b, we propose a new graph and accept it if it is acyclic.

Algorithm B

B1 Draw $U \sim \mathrm{Uniform}(0, 1)$.

B2 If $U \leq 1 - b$, then return G.

B3 Draw a node pair ij with probability b_{ij}/b.

B4 If G^{ij} is acyclic, then return G^{ij}; else return G.

[3]Unlike Giudici and Castelo (2003), here we prefer to also include reversals that do not change the Markov equivalence class.

Observe that this is a MH algorithm with the proposal distribution $Q(G^{ij}|G) = b_{ij}$ for any acyclic $G^{ij} \neq G$. The untruncated acceptance ratio $R(G^{ij}, G)$ thus equals 1, since $\beta(G, G^{ij})\pi(G^{ij}) = \beta(G^{ij}, G)\pi(G)$. It follows that π is the stationary distribution of the chain, as desired. (See Suppl. A.1 for a more detailed proof.)

When b is small, we obtain further expeditation by simulating *multiple consecutive steps* by drawing a *single* geometric random variable with mean $1/b$ (our present implementation), or by attaching each state with a weight equalling $1/b$, similar to a related birth-death process formulation for sampling undirected networks (Mohammadi and Wit, 2015). In contrast, for Gibbs sampling it is desirable to lower self transition probabilities (Neal, 2024).

3.2 FAST GENERATION OF PROPOSALS

For drawing node pairs ij with probabilities b_{ij}/b, we partition the pairs into groups $N_j = \{ij : i \neq j\}$, each with its partial sum $b_j = \sum_i b_{ij}$. We draw a pair ij in two phases: first j with probability b_j/b, and then i from N_j with probability b_{ij}/b_j.

Using a *binary sum-tree* data structure (described in Suppl. A.2) for the n distributions $(b_{ij}/b_j)_i$ and the distribution $(b_j/b)_j$, generating ij takes only $O(\log n)$ time. A binary sum-tree stores nonnegative reals $a_1, \ldots, a_n$ in the leaves of a balanced rooted binary tree, letting each inner node store the sum of the values of its children. We can draw u with probability proportional to a_u by first drawing a random variable r from the uniform distribution on the range $[0, \sum_j a_j]$, and then, starting from the root, iteratively selecting the left of right child in a binary-search fashion to locate the smallest u for which $\sum_{j=1}^{u} a_j \geq r$.

The motivation for the partitioning stems from the need to efficiently update the distribution after adding, removing, or reversing an arc. Notably, an update induced by the node pair ij can only modify the parent sets of i and j. Thus, it only affects the probabilities b_{uv} in which either u or v, or both, belong to $\{i, j\}$. After a removal the update is as below; after an addition it is identical; after a reversal we first remove the arc and then add the reversed arc.

Algorithm U (Update after removing arc ij)

U1 Recompute b_{ij} and b_{ji}.

U2 Recompute b_{uj} for all $u \neq i, j$.

U3 Recompute b_{jv} for all parents $v \neq i$ of j.

U4 Recompute the affected marginals b_j, b_v, and b.

Steps U1 and U2 are straightforward to compute in time $O(n)$, and step U3 in time $O(\ell_j)$, where ℓ_j is the number of parents of j. In step U4, we construct binary sum-trees for the distributions $(b_{uj}/b_j)_u$ and $(b_u/b)_u$ in time $O(n)$.

Table 1: Speed of *Gibby* vs. *GC* on datasets of size 1 000 sampled from benchmark Bayesian networks

				Steps per microsecond			Acceptance ratio (%)	
Network	Nodes	Arcs	Max-indeg	*GC*	*Gibby*†	*Gibby*‡	a	b
ALARM	37	46	4	3.0	**67**	38	0.50	4.6
HAILFINDER	56	66	4	3.1	**91**	83	0.31	1.2
PATHFINDER	109	195	5	1.9	18	**19**	0.34	1.3
ANDES	223	338	6	2.0	46	**58**	0.058	0.19
PIGS	441	592	2	3.1	553	**582**	0.017	0.14

†Constant-time acyclicity checks by maintaining the ancestor relation ‡Acyclicity checks by path-finding algorithms

Furthermore, we update the binary sum-trees for the distributions $(b_{uv}/b_v)_u$ for each of the ℓ_j updates of b_{jv}, each in time $O(\log n)$.

3.3 FAST ACYCLICITY CHECKS

Recall that the *GC* algorithm of Giudici and Castelo (2003) maintains the ancestor relation to enable constant-time acyclicity checks. This is motivated by the low acceptance rate of the algorithm: one expects a large number of acyclicity checks per accepted move.

But *Gibby* proposes a new graph with probability b, and the acceptance rate is substantially higher because only cyclic proposals are rejected. This shifts the optimal tradeoff between the work per proposed graph and the work per accepted move. While sophisticated methods are known for balancing the query and update time (Hanauer et al., 2020), we have found that simple path-finding algorithms (such as depth-first or breadth-first search) are sufficiently fast to not dominate the overall computation time. The worst-case complexity is linear in the number of arcs, $O(\ell)$. This should be contrasted with *GC*'s $O(n\ell)$ complexity per update (after arc removal).

As noted by the original authors, however, the hidden constant factor is small for *GC*, thanks to 64-bit processors being able to handle 64 nodes by a single operation. Modern vector processing capabilities (SIMD instructions) may boost this further. Acknowledging this, we have also implemented a version of *Gibby* that takes the approach of *GC* for acyclicity checking.

3.4 PERFORMANCE IN PRACTICE

We compared our C++ implementations of *GC* and *Gibby* on datasets drawn from five benchmark BNs. We used the sparse prior with the maximum-indegree of the data-generating network. Local scores were precomputed for parent sets of sizes 0 and 1; for larger parent sets, local scores were computed on demand and cached. For each dataset, we ran both *GC* and *Gibby* for 2×10^{10} steps, gauging the running time only for the latter half to lessen the influence of the demanding score computations in the burn-in phase of the simulation. As *GC* and *Gibby* simulate exactly the same Markov chain, they differ only in speed, the mixing properties being the same. Table 1 shows results on datasets of size 1 000 (see Suppl. E.1 for sizes 500 and 2 000).

Our implementation of *GC* takes 2–3 million steps per second, across the tested datasets, making *billion*-step simulations computationally feasible. For comparison, in the literature (Giudici and Castelo, 2003; Grzegorczyk, 2023, 2024; Grzegorczyk and Husmeier, 2008; Su and Borsuk, 2016) the reported simulations are in a few million steps. The larger the network, the lower the acceptance rate (value a) and the less frequent updating of the data structures. While the running time per step is about the same, the number of accepted moves is significantly smaller for larger networks.

Gibby, in contrast, takes advantage of low acceptance rate, reflected in the tentative acceptance rate (value b). On the largest of the tested networks, *Gibby* is about 200 times faster than *GC*, rendering *trillions* of steps computationally feasible; also on the smaller networks, we observe at least an order-of-magnitude speedup. Of the two alternative ways to manage acyclicity, maintaining the ancestor relation has a clear advantage on the smaller networks, while calling a path-finding algorithm appears to be faster on the larger networks. This can be explained by the time complexity of updating the ancestor relation: even if updates are needed less frequently on larger networks, the update cost grows relatively fast with the number of nodes.

4 SCORE PRUNING

The REV and MBR moves are computationally more expensive than the basic moves because they require summing local scores over possible parent sets for one or multiple nodes. Given a node $i \in N$ and two sets $T \subseteq U \subseteq N \setminus \{i\}$, we need the *interval sum*

$$\pi_i[T, U] := \sum_{T \subseteq S \subseteq U} \pi_i(S)\,.$$

For REV and MBR, the lower set T is either empty or a singleton set. To answer such interval queries (i, T, U), a

common approach is to loop over *all* possible parent sets S of i in decreasing order by the scores, accumulate the sum by $\pi_i(S)$ if $T \subseteq S \subseteq U$, and stop as soon as the contributions of the remaining scores are guaranteed to be only negligible (Friedman and Koller, 2003; Niinimäki and Koivisto, 2013; Niinimäki et al., 2016; Viinikka et al., 2020).

We next present a way to shorten the list of local scores by safely discarding parent sets that are dominated by other sets in the list. Our pruning method is inspired by the folklore pruning rule (Teyssier and Koller, 2005) that applies when one seeks a DAG that *maximizes* the score; then one can discard set S if it has a subset R with an equal or better score, that is, $\pi_i(S) \leq \pi_i(R)$. Our pruning rule is similar in spirit but technically more involved, as we have to consider approximate evaluations of interval queries.

4.1 THEORY

Let us focus on a fixed child node i. For brevity, write V for the set $N \setminus \{i\}$ and f for the local score function π_i. Recall that the local scores are unnormalized probabilities and thus nonnegative numbers. We formulate the goal of approximation as a guarantee in the relative error.

Definition 1 (ϵ-close)**.** *Let V be a finite set and $\epsilon \geq 0$. We say that $\tilde{f} : 2^V \to \mathbb{R}_{\geq 0}$ is* ϵ-close *to $f : 2^V \to \mathbb{R}_{\geq 0}$ if*

$$(1-\epsilon) f[T,U] \leq \tilde{f}[T,U] \leq f[T,U]$$

for all $T, U \subseteq V$ with $|T| \leq 1$.

Here, we only allow underestimation of the exact sum. This is motivated by our pruning method that forms $\tilde{f}$ by simply discarding some sets S, i.e., setting $\tilde{f}(S)$ to 0.

Definition 2 (ϵ-pruning)**.** *Let V be a finite set of K elements and $\epsilon \geq 0$. Let $f : 2^V \to \mathbb{R}_{\geq 0}$. The* ϵ-pruning *of f is the function $\tilde{f} : 2^V \to \mathbb{R}_{\geq 0}$ obtained as*

$$\tilde{f}(S) := \begin{cases} 0\,, & \text{if } f(S) < \epsilon \cdot \psi(j,S) \text{ for all } j \in S, \\ f(S)\,, & \text{otherwise}, \end{cases}$$

where

$$\psi(j,S) = \sum_{j \in R \subseteq S} f(R) \big(1 + 1/K\big)^{|R|-K} K^{|R|-|S|}\,.$$

In words, discard S if its score is much less than sums of appropriately weighted scores of its subsets.

Theorem 1. *The ϵ-pruning of f is ϵ-close to f.*

Proof. We prove the statement in a more general form, namely with

$$\psi(j,S) = \sum_{j \in R \subseteq S} f(R)\, w(R,S)\,,$$

only assuming that

$$\sum_{R \subseteq S \subseteq U} w(R,S) \leq 1 \quad \text{for all } R \subseteq U \subseteq V\,. \tag{2}$$

Observe that this inequality holds for the function $w(R,S) = (1+1/K)^{|R|-K} K^{|R|-|S|}$, the interval sum then equalling $(1+1/K)^{|U|-K} \leq 1$ by the binomial theorem.

Let $T \subseteq U \subseteq V$. We bound the difference

$$\begin{aligned}
&\sum_{T \subseteq S \subseteq U} f(S) - \sum_{T \subseteq S \subseteq U} \tilde{f}(S) && (3)\\
&\quad\leq \sum_{T \subseteq S \subseteq U} \min_{j \in S} \big\{\epsilon \cdot \psi(j,S)\big\} && (4)\\
&\quad\leq \epsilon \cdot \sum_{T \subseteq S \subseteq U} \sum_{T \subseteq R \subseteq S} f(R)\, w(R,S) && (5)\\
&\quad= \epsilon \cdot \sum_{T \subseteq R \subseteq U} \sum_{R \subseteq S \subseteq U} f(R)\, w(R,S) && (6)\\
&\quad\leq \epsilon \cdot \sum_{T \subseteq R \subseteq U} f(R)\,, && (7)
\end{aligned}$$

proving the claim. Justifications for each step above: (4) For each set S, the difference is 0 or at most $\epsilon \cdot \psi(j,S)$ for $j \in S$. (5) If $T = \emptyset$, the inner sum is without the constraint $j \in S$. Otherwise, $T = \{j\}$ for some $j \in S$. (6) Switch the order of summation. (7) Use the assumption (2). □

This per-node guarantee is sufficient for ensuring that the induced approximate distribution over DAGs is close to the exact distribution; we just have to scale the error parameter by the number of nodes n. Indeed. we can show (Suppl. Theorem 2) that with (ϵ/n)-pruning of π_i for all nodes i, the Kullback–Leibler divergence and the total variation distance of the resulting approximate posterior from the exact one are at most $\epsilon/(1-\epsilon)$ and ϵ, respectively.

4.2 PRACTICE

The ϵ-pruning $\tilde{\pi}_i$ of a score function π_i is straightforward to compute: First, $\pi_i(S)$ is computed for all possible parent sets S. Then, by a second pass, each set S is either pruned or kept depending on whether $\pi_i(S)$ is less than $\epsilon \cdot \psi_i(j,S)$ for all $j \in S$; here ψ_i is defined as ψ but, of course, separately for each node i. For fixed i and j, the evaluation of $\psi_i(j,S)$ takes time $O\big(2^{|S|}\big)$. We call this *complete* pruning, since it computes the scores also for all sets that get pruned.

Ideally, one would only score the sets that will be kept. When pruning significantly reduces the number of kept sets, such lazy computation could be much faster than complete pruning. We have partially implemented this idea in what we call *bottom-up* pruning, as follows. We visit possible parent sets S in increasing order by size. As before, we prune S if $\pi_i(S)$ is less than $\epsilon \cdot \psi_i(j,S)$ for all $j \in S$. However, now we also prune S if *all* its subsets $S \setminus \{j\}$ for $j \in S$ were

Table 2: Efficiency of pruning on datasets of size 1 000 sampled from benchmark Bayesian networks

				$\epsilon = 2^{-15}$			$\epsilon = 2^{-10}$		
Network	Nodes	Max-indeg	$K^\dagger$	Kept (%)	$T_\mathrm{b}{}^*$	T_c	Kept (%)	T_b	T_c
ALARM	37	4	36	5.9	8.9	15	3.7	7.8	15
HAILFINDER	56	4	55	0.090	6.5	127	0.087	6.4	127
PATHFINDER	109	5	64	22	9230	17800	15	8450	17100
ANDES	223	6	64	0.16	10300	–	0.041	3470	–
PIGS	441	2	64	18	11	11	16	11	11
PIGS	441	$4^\ddagger$	64	0.11	125	5490	0.10	116	5640

†No. candidate parents per node *No. seconds by bottom-up (T_b) and complete pruning (T_c) ‡Larger than in the generating network

pruned. Thus, for example, if all sets of size 5 are pruned, there is no need to score sets of size 6 or larger. While we cannot prove that this heuristic never prunes any extra set, we have empirically observed that complete and bottom-up pruning typically yield exactly the same result.

With hundreds of nodes and relatively large maximum-indegree bound, say 5 or 6, further heuristics may be needed for feasibility, both concerning the time requirement of score computations and the memory requirement for storing the scores. To this end, we adopt the idea of preselecting some number K of candidate parents per node (Friedman and Koller, 2003; Kuipers et al., 2022; Viinikka et al., 2020); as the candidate parents of node i, we simply selected the K nodes j with the largest scores $\pi_i(\{j\})$ (Friedman and Koller, 2003), but also other heuristics could be considered (Viinikka et al., 2020), e.g., through constraint-based methods (Kuipers et al., 2022). In addition, we also consider parent sets not contained in the set of candidates, however, subject to a lower maximum-indegree bound, which we set to roughly match the amount of available memory.

Table 2 shows empirical results on datasets of size 1 000 generated from five benchmark BNs (see Suppl. E.2 for sizes 500 and 2 000). We observe that the efficiency of pruning varies in scale across the datasets and the variability is not easily explained by the parameters of the generating networks, such as the number of nodes or arcs, or max-indegree. That said, a larger max-indegree raises chances for significant pruning to take place; this holds especially when we use a maximum-indegree larger than the actual value of the generating networks (second instance of PIGS).

Aligned with our expectations, bottom-up pruning is much faster than complete pruning when the fraction of parent sets kept after pruning is small. The results appear to be insensitive to the relative error parameter ϵ. The only exception is ANDES, for which increasing ϵ from 2^{-15} to 2^{-10} reduced the kept parents to about one fourth and the running time of bottom-up pruning to about one third.

5 COMPARISON TO OTHER SAMPLERS

We compared our algorithm *Gibby* to two recent Bayesian DAG samplers, *BiDAG* (Suter et al., 2023) and *DAG-GFlowNet* (Deleu et al., 2022), both of which can handle categorical data and support modular (e.g., uniform) structure priors. We excluded from the comparison other samplers that cannot express these models (see Suppl. B for a short review of related methods).

BiDAG is a sophisticated implementation of the partition MCMC method (Kuipers and Moffa, 2017). It simulates a Markov chain on ordered node partitions, each corresponding to exponentially many DAGs. Various ideas are employed to find a small set of candidate parents for each node, to enable fast access to sums of local scores (similar to the interval sums of Sect. 4) using precomputed look-up tables. We experimented with two variants: *BiDAG full* explores the full DAG state space without any restrictions; *BiDAG iter* alternates between inference in a restricted parent-set space and posterior-guided expansion of that space.

DAG-GFlowNet samples DAGs using generative flow networks in a fashion similar to sequential importance sampling. It first learns a "transition probability distribution" for moving from a given DAG to another DAG with one additional arc. Then it can generate DAGs independently from an approximate posterior.

For all the samplers, we used the sparse structure prior and the Dirichlet priors for the parameters with the equivalent sample size 1.0. The maximum-indegree parameter was set separately for each dataset, as described in Suppl. E.3. For more details on our the choices for the user parameters and on the computing environment, see Suppl. B and C.

In addition, in Suppl. E.4, we present experiments on continuous data using the BGe score for linear Gaussian models. We include *DiBS* (Lorch et al., 2021) in these experiments, as it is a variational Bayesian structure learning method compatible with the BGe score. Nonetheless, our results indicate that *DiBS* struggles to approximate the posterior distribution of DAGs, even for small, low-dimensional datasets. We therefore omit this comparison from the main body of the

paper. Additional details on *DiBS* are given in Suppl. B.

5.1 SMALL NETWORKS

We ran the three samplers on datasets of size 1 000 sampled from selected small benchmark BNs (Scutari, 2022), as well as on the ZOO benchmark dataset of size 101 (Forsyth, 1990). *BiDAG full* was restricted to ASIA and SACHS, since it is only feasible for small DAGs. We measure the accuracy of a sample of DAGs by the *MAD*, i.e., the *maximum absolute deviation* $|\hat{p}_{ij} - p_{ij}|$ over all node pairs ij. Here $\hat{p}_{ij}$ is the relative frequency of arc ij in the sampled DAGs and p_{ij} is the respective exact probability we compute by an exponential-time algorithm (Harviainen and Koivisto, 2024; Pensar et al., 2020; Tian and He, 2009) (see Suppl. A.4).

We observe (Fig. 1) that the arc posterior probability estimates by *Gibby* rapidly become near-optimal, the MAD decaying roughly at the rate $O\big(1/\sqrt{t}\big)$ in the sample size t. The estimates by *BiDAG* improve as t increases for ASIA, with the *full* variant performing slightly better. For the other networks, the MAD stays at high values, suggesting that the Markov chain does not mix well; further evidence for this is given in Suppl. E.3. *DAG-GFlowNet* samples DAGs only after a training phase, which lasted approximately 30 minutes for both ASIA and SACHS. It performs reasonably in some runs on ASIA, but more generally it does not produce reliable output for the tested datasets; as the approach of *DAG-GFlowNet* is very different from that of *Gibby* and *BiDAG*, we defer the detailed results to Suppl. E.3.

5.2 LARGE NETWORKS

On large networks, we only compare *Gibby* to *BiDAG iter*, given the weak performance of *DAG-GFlowNet*. We generated datasets of size 1 000 from two benchmark BNs and visually inspect the traceplots of the samplers (Fig. 2). We compare the sampled DAGs to the data-generating DAG G^* in terms of the posterior probability, but not with respect to any structural metrics, as the task is to approximate the posterior distribution (usually not well represented by G^*).

We observe that both samplers quickly reach regions of DAGs whose score exceeds that of the data-generating DAG. *Gibby* appears to mix well, the five independent runs showing very similar behavior. *BiDAG iter* is less robust, different runs often getting stuck at different local modes. The arc posterior probability estimates (Suppl. E.3, Fig. 7) are highly correlated between the *Gibby* runs, whereas different *BiDAG* runs yield largely varying estimates.

6 CONCLUDING REMARKS

We have advanced computational methods for approximate posterior sampling of DAGs. Our results suggest that currently the best methods, scaling beyond exact algorithms, are based on Markov chains with moves that resample entire parent sets of one or multiple nodes. However, the computational cost of such moves increases rapidly with the user-defined maximum indegree, which has limited their practical use to small DAGs or modest indegree bounds.

To mitigate this, we introduced a pruning rule that can significantly reduce precomputing time, lower memory requirements, and speed up per-move computations. While pruning is highly effective in many cases, it can be less efficient on certain datasets, necessitating additional techniques or constraints to ensure computational feasibility. In such scenarios, our enhancement for simulating single-arc moves should prove particularly advantageous.

There are several avenues for further research. First, our score pruning technique can significantly reduce the computational burden of moves that sample entire parent sets according to their score. This encourages the exploration of new moves, similar to REV and MBR, that could further expand the set of DAGs reachable from a given DAG, and thereby expedite mixing and convergence to the posterior.

Another direction is to adopt the new techniques in the partition MCMC framework (Kuipers and Moffa, 2017; Kuipers et al., 2022). The techniques could address some current weaknesses of *BiDAG*, namely frequent self-transitions in the space of node partitions and the lack of moves based on resampling parent sets.

Finally, we believe our ideas to expedite basic moves (upper bounding the acceptance ratio and simulating a simple geometric random variable) and to prune local scores (weighted domination to preserve a family of approximate subset-sums) are applicable more broadly. They are likely to find applications also in other related contexts of Markov chain simulation and approximate likelihood computations.

Acknowledgements

The research was partially supported by the Research Council of Finland, grant 351156.

References

Yashas Annadani, Nick Pawlowski, Joel Jennings, Stefan Bauer, Cheng Zhang, and Wenbo Gong. BayesDAG: Gradient-based posterior inference for causal discovery. In *Advances in Neural Information Processing Systems, NeurIPS 2023*, volume 36, pages 1738–1763, 2023.

Mark Bartlett and James Cussens. Integer Linear Programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

Wray L. Buntine. Theory refinement on Bayesian networks. In *The Seventh Annual Conference on Uncertainty in*

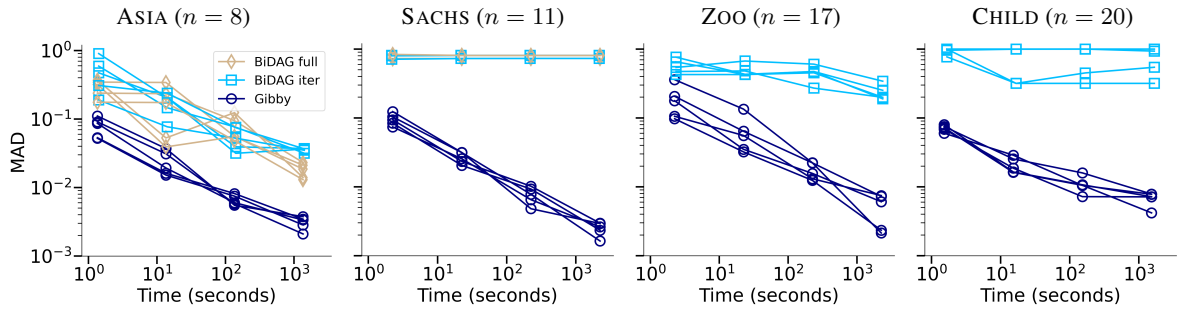

Figure 1: Performance on small networks. For each sampler, shown is the MAD of five independent runs as a function of running time. For each run, 100 000 DAGs were collected with even spacing.

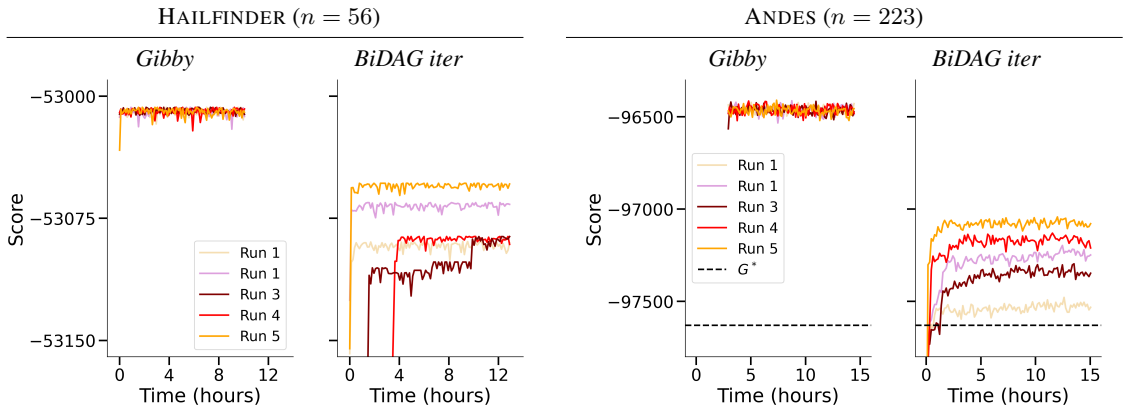

Figure 2: Performance on large networks. For *Gibby* the number of candidate parents K were set as in Table 2. For each sampler, shown is the log posterior probability of the sampled DAGs in five independent runs. For HAILFINDER, the score of the data-generating DAG G^* is around -56010 and not visible. For ANDES, *Gibby* spends the first hours in score computations, before sampling.

Artificial Intelligence, UAI 1991, pages 52–60. Morgan Kaufmann, 1991.

Gregory F. Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Mach. Learn.*, 9:309–347, 1992.

Chris Cundy, Aditya Grover, and Stefano Ermon. BCD Nets: Scalable variational approaches for Bayesian causal discovery. In *Advances in Neural Information Processing Systems, NeurIPS 2021*, volume 34, pages 7095–7110, 2021.

James Cussens. GOBNILP: Learning Bayesian network structure with integer programming. In *The 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 605–608. PMLR, 2020.

Tristan Deleu, António Góis, Chris Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *The Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022*, volume 180 of *Proceedings of Machine Learning Research*, pages 518–528. PMLR, 2022.

Tristan Deleu, Mizu Nishikawa-Toomey, Jithendaraa Subramanian, Nikolay Malkin, Laurent Charlin, and Yoshua Bengio. Joint Bayesian inference of graphical structure and parameters with a single generative flow network. In *Advances in Neural Information Processing Systems, NeurIPS 2023*, volume 36, pages 31204–31231, 2023.

Ralf Eggeling, Jussi Viinikka, Aleksis Vuoksenmaa, and Mikko Koivisto. On structure priors for learning Bayesian networks. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019*, volume 89 of *Proceedings of Machine Learning Research*, pages 1687–1695. PMLR, 2019.

Richard Forsyth. Zoo. UCI Machine Learning Reposi-

tory, 1990. URL https://doi.org/10.24432/C5R59V.

Nir Friedman and Daphne Koller. Being Bayesian about network structure. A Bayesian approach to structure discovery in Bayesian networks. *Mach. Learn.*, 50:95–125, 2003.

Dan Geiger and David Heckerman. Learning Gaussian networks. In *The Tenth International Conference on Uncertainty in Artificial Intelligence, UAI 1994*, pages 235–243. Morgan Kaufmann Publishers Inc., 1994.

Paolo Giudici and Robert Castelo. Improving Markov chain Monte Carlo model search for data mining. *Mach. Learn.*, 50:127–158, 2003.

Robert J. B. Goudie and Sach Mukherjee. A Gibbs sampler for learning DAGs. *J. Mach. Learn. Res.*, 17:1–39, 2016.

Marco Grzegorczyk. Being Bayesian about learning Gaussian Bayesian networks from incomplete data. *Int. J. Approx. Reason.*, 160:108954, 2023.

Marco Grzegorczyk. Being Bayesian about learning Bayesian networks from ordinal data. *Int. J. Approx. Reason.*, 170:109205, 2024.

Marco Grzegorczyk and Dirk Husmeier. Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Mach. Learn.*, 71: 265–305, 2008.

Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Faster fully dynamic transitive closure in practice. In *18th International Symposium on Experimental Algorithms, SEA 2020*, volume 160 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Juha Harviainen and Mikko Koivisto. Faster perfect sampling of Bayesian network structures. In *The 40th Conference on Uncertainty in Artificial Intelligence, UAI 2024*, volume 244 of *Proceedings of Machine Learning Research*, pages 1558–1568. PMLR, 2024.

Wilfred K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57: 97–109, 1970.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Mach. Learn.*, 20:197–243, 1995.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI Machine Learning Repository, 2023. URL https://archive.ics.uci.edu.

Neville Kenneth Kitson, Anthony C. Constantinou, Zhigao Guo, Yang Liu, and Kiattikun Chobtham. A survey of Bayesian network structure learning. *Artif. Intell. Rev.*, 56:8721–8814, 2023.

Mikko Koivisto and Kismat Sood. Exact Bayesian structure discovery in Bayesian networks. *J. Mach. Learn. Res.*, 5: 549–573, 2004.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

Jack Kuipers and Giusi Moffa. Partition MCMC for inference on acyclic digraphs. *J. Am. Stat. Assoc.*, 112: 282–299, 2017.

Jack Kuipers, Giusi Moffa, and David Heckerman. Addendum on the scoring of Gaussian directed acyclic graphical models. *The Annals of Statistics*, 42(4):1689–1691, 2014.

Jack Kuipers, Polina Suter, and Giusi Moffa. Efficient sampling and structure learning of Bayesian networks. *J. Comput. Graph. Stat.*, 31:639–650, 2022.

David A. Levin and Yuval Peres. *Markov Chains and Mixing Times*, volume 107. American Mathematical Soc., 2017.

Lars Lorch, Jonas Rothfuss, Bernhard Schölkopf, and Andreas Krause. DiBS: Differentiable Bayesian structure learning. In *Advances in Neural Information Processing Systems, NeurIPS 2021*, volume 34, pages 24111–24123, 2021.

David Madigan and Jeremy York. Bayesian graphical models for discrete data. *Int. Stat. Rev.*, 63:215–232, 1995.

Abdolreza Mohammadi and Ernst C. Wit. Bayesian structure learning in sparse Gaussian graphical models. *Bayesian Analysis*, 10(1):109 – 138, 2015.

Radford M. Neal. Modifying Gibbs sampling to avoid self transitions, 2024. arXiv:2403.18054 [stat.CO].

Teppo Niinimäki and Mikko Koivisto. Treedy: A heuristic for counting and sampling subsets. In *The Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013*, pages 469–477. AUAI Press, 2013.

Teppo Niinimäki, Pekka Parviainen, and Mikko Koivisto. Structure discovery in Bayesian networks by sampling partial orders. *J. Mach. Learn. Res.*, 17:1–47, 2016.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2009.

Johan Pensar, Topi Talvitie, Antti Hyttinen, and Mikko Koivisto. A Bayesian approach for estimating causal effects from observational data. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, volume 34, pages 5395–5402. AAAI Press, 2020.

Felix L. Rios, Giusi Moffa, and Jack Kuipers. Benchpress: A versatile platform for structure learning in causal and probabilistic graphical models. *Journal of Statistical Software*, 114(12):1–43, 2025.

Marco Scutari. Bayesian Network Repository, 2022. URL `www.bnlearn.com/bnrepository`. Accessed: 2025-04-25.

Chengwei Su and Mark E. Borsuk. Improving structure MCMC for Bayesian networks through Markov blanket resampling. *J. Mach. Learn. Res.*, 17:118:1–118:20, 2016.

Polina Suter, Jack Kuipers, Giusi Moffa, and Niko Beerenwinkel. Bayesian structure learning and sampling of Bayesian networks with the *R* package BiDAG. *Journal of Statistical Software*, 105(9):1–31, 2023.

Topi Talvitie, Aleksis Vuoksenmaa, and Mikko Koivisto. Exact sampling of directed acyclic graphs from modular distributions. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 965–974. PMLR, 2020.

Marc Teyssier and Daphne Koller. Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *The 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pages 540–549. AUAI Press, 2005.

Jin Tian and Ru He. Computing posterior probabilities of structural features in Bayesian networks. In *The Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009*, pages 538–547. AUAI Press, 2009.

Jussi Viinikka and Mikko Koivisto. Layering-MCMC for structure learning in Bayesian networks. In *The Thirty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2020*, volume 124 of *Proceedings of Machine Learning Research*, pages 839–848. AUAI Press, 2020.

Jussi Viinikka, Antti Hyttinen, Johan Pensar, and Mikko Koivisto. Towards scalable Bayesian learning of causal dags. In *Advances in Neural Information Processing Systems, NeurIPS 2020*, volume 33, pages 6584–6594, 2020.

Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: DAG structure learning with graph neural networks. In *The 36th International Conference on Machine Learning, ICML 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 7154–7163. PMLR, 2019.

Scaling Up Bayesian DAG Sampling (Supplementary Material)

Daniele Nikzad[1] **Alexander Zhilkin**[1] **Juha Harviainen**[1] **Jack Kuipers**[2] **Giusi Moffa**[3] **Mikko Koivisto**[1]

[1]University of Helsinki
[2]ETH Zurich
[3]University of Basel

A ALGORITHMS AND DATA STRUCTURES

The section proves the convergence of the chain with fast basic moves, describes the binary sum-tree data structure, proves Theorem 1, and gives bounds for the posterior mass lost in score pruning.

A.1 CONVERGENCE OF FAST BASIC MOVES

We prove that the Markov chain constructed in Section 3 converges to the posterior $P(G|X) = \pi(G)/Z$, where Z is a normalizing constant. To this end, it is sufficient to show that the chain is irreducible and aperiodic, and that the posterior is a stationary distribution of the chain (Levin and Peres, 2017, Theorem 4.9, for example). We make the following mild assumptions:

$$\pi(G) > 0 \text{ for all DAGs } G \text{ that contain exactly one arc}\,, \tag{8}$$

$$\text{if } \pi(G) > 0, \text{ then } G \text{ has an arc removing of which results in a DAG } G' \text{ with } \pi(G') > 0\,, \tag{9}$$

$$q_{ij} > 0 \text{ for all node pairs } ij\,, \tag{10}$$

$$q_{ij} = q_{ji} \text{ for all node pairs } ij\,. \tag{11}$$

Denote by $p(G^{ij}|G)$ the transition probability of the chain of moving in one step from DAG G to DAG G^{ij} obtained from G by adding, removing, or reversing arc ij. Recall that, by Algorithm B, we have

$$p(G^{ij}|G) = b \cdot \frac{b_{ij}}{b} = q_{ij} \min\left\{1, \frac{\pi^*(G^{ij})}{\pi(G)}\right\} = q_{ij} \min\left\{1, \frac{\pi(G^{ij})}{\pi(G)}\right\}. \tag{12}$$

Now, to see that the chain is irreducible, consider two arbitrary DAGs G and G''. From G, we can reach the empty DAG by a sequence of arc removals, and from the empty DAG, we can reach G'' by a sequence of arc additions. Thus G and G'' are connected by single-arc moves, each of which is taken with a positive transition probability (12) by assumption (9).

To see that the chain is aperiodic, it suffices to show that the chain can return from the empty DAG to the empty DAG in both 2 and 3 steps. For example, we could add arc ij and remove it, or we could add ij, reverse ij, and remove ji. By assumptions (8)–(10) and formula (12), these steps have positive transition probabilities.

Finally, to see that π/Z is a stationary distribution of the chain, it suffices to show that the following detailed balance condition holds (Levin and Peres, 2017, Prop. 1.20, for example):

$$\pi(G)p(G'|G) = \pi(G')p(G|G') \quad \text{for all DAGs } G \text{ and } G'\,.$$

If $G = G'$, the condition is trivial. Assume $G' = G^{ij}$ with some ij, for otherwise the transition probabilities vanish on both sides and the condition trivially holds. If $\pi(G) = 0$, then, by (12), we have $p(G|G^{ij}) = 0$, and thus the condition holds. By

symmetry, we may now assume that $\pi(G)$ and $\pi(G^{ij})$ are nonzero. Using (12), we get

$$\pi(G)p(G^{ij}|G) = q_{ij} \min\left\{\pi(G), \pi(G^{ij})\right\}.$$

Now, if G^{ij} is obtained by adding or removing ij, then we also have

$$\pi(G^{ij})p(G|G^{ij}) = q_{ij} \min\left\{\pi(G^{ij}), \pi(G)\right\},$$

and thus the condition holds. If G^{ij} is obtained by reversing arc ji to ij, then instead

$$\pi(G^{ij})p(G|G^{ij}) = q_{ji} \min\left\{\pi(G^{ij}), \pi(G)\right\},$$

and the condition holds by assumption (11).

A.2 BINARY SUM-TREE

A binary sum-tree implements three operations:

`Initialize`(a) initializes the data structure with a vector $a = (a_1, \ldots, a_n)$ of nonnegative reals.

`Replace`(i, x) replaces the ith value a_i by x.

`Draw`$()$ returns a draw from the categorical distribution $(p_1, \ldots, p_n)$, where $p_i = a_i / \sum_j a_j$.

The data structure is a balanced rooted binary tree with n leaves. If n is originally not a power of two, we pad the vector a with a sufficient number of zeros.

Each node in the tree stores a value. The ith leaf stores a_i. The root and the internal nodes store the sum of the values stored by their two children.

The data structure can be efficiently implemented using an array A of $2n$ values (floating point numbers). The position of the ith leaf in A is $n + i$. The position of the parent of i is $\lfloor (n + i)/2 \rfloor$.

The initialization operation first fills in the last n positions of A by the input values $a_1, \ldots, a_n$. Then it computes the values of all inner nodes and the root in a bottom-up fashion. This takes $O(n)$ time.

Replacing a_i by x requires a constant-time update for each node from the path from the leaf to the root, thus taking time $O(\log n)$ in total.

The draw operation can be implemented in time $O(\log n)$ as follows. First, a random number r is drawn from the range $[0, s]$, where s is the value of the root. Then, starting from the root, the left child is selected if r is less or equal to its value ℓ; otherwise the right child is selected and r is updated to $r - \ell$. The previous step is repeated until a leaf node is reached. The value of the leaf is returned.

A.3 BOUNDING THE POSTERIOR MASS LOST DUE TO PRUNING

Theorem 2. *Let $\epsilon \in (0, 1)$. For each node $i = 1, \ldots, n$, let $\tilde{\pi}_i$ be an (ϵ/n)-pruning of π_i. For any DAG on the node set $\{1, \ldots, n\}$ with arc set A, let $\pi(A) = \prod_i \pi(A_i)$ and $\tilde{\pi}(A) = \prod_i \tilde{\pi}_i(A_i)$. Let $Z = \sum_A \pi(A)$ and $\tilde{Z} = \sum_A \tilde{\pi}(A)$, where the sums are over all DAGs on the n nodes. Then*

- **(i)** *$\tilde{Z} \geq (1 - \epsilon)Z$;*
- **(ii)** *The Kullback–Leibler divergence of $\tilde{\pi}/Z$ from π/Z is at most $\epsilon/(1 - \epsilon)$;*
- **(iii)** *The total variation distance between π/Z and $\tilde{\pi}/\tilde{Z}$ is at most ϵ.*

Remark 1. Note that the Kullback–Leibler divergence of π/Z from $\tilde{\pi}/Z$ is undefined (or ∞), unless there is no pruning, i.e., $\pi = \tilde{\pi}$. When π/Z is considered a "ground truth" this variant is sometimes called the *forward KL divergence*, while the variant in claim (ii) is called the *reverse KL divergence*.

For a proof, we first extend the approximation guarantee of interval sums to what we call intersection sums.

Proposition 3. *For a function f from the subsets of a finite set V to nonnegative real numbers, write*

$$f(T;U) := \sum_{S \subseteq U : S \cap T \neq \emptyset} f(S)\,, \quad \textit{for } T \subseteq U \subseteq V\,.$$

Suppose $\tilde{f}$ is δ-close to f with $\delta > 0$. Then $\tilde{f}(T;U) \geq (1-\delta) f(T;U)$ for all $T \subseteq U \subseteq V$.

Proof. Label the elements of T arbitrarily as $t_1, \ldots, t_\ell$. Write in terms of interval sums:

$$f(T;U) = \sum_{j=1}^{\ell} f\big[\{t_j\}, U \setminus \{t_1, \ldots, t_{j-1}\}\big]\,.$$

Now the claim follows because $\tilde{f}[T', U'] \geq (1-\delta) f[T', U']$ for each term in the sum. □

Armed with the above result, we turn to prove claim (i). By Equation 3 in Kuipers and Moffa (2017), and also Section 3.1 in Viinikka et al. (2020), we can write

$$Z = \sum_A \pi(A) = \sum_R \prod_{j=1}^{k} \prod_{i \in R_i} \sum_{\substack{A_i \subseteq R_1 \cup \cdots \cup R_{j-1} \\ A_i \cap R_{j-1} \neq \emptyset}} \pi_i(A_i)\,,$$

where the sum is over all ordered set partitions $R = (R_1, \ldots, R_k)$ of $\{1, \ldots, n\}$. By writing $\tilde{Z}$ correspondingly and using the assumption that each $\tilde{\pi}_i$ is (ϵ/n)-close to π_i, Proposition 3 gives that each sum of $\tilde{\pi}_i(A_i)$ is at least $1 - \epsilon/n$ times the corresponding sum of $\pi_i(A_i)$. Thus

$$\tilde{Z} \geq (1 - \epsilon/n)^n Z \geq (1 - \epsilon) Z\,.$$

To prove claim (ii), write the Kullback–Leibler divergence as

$$\sum_{A : \tilde{\pi}(A) > 0} \tilde{\pi}(A)/\tilde{Z} \cdot \ln \frac{\tilde{\pi}(A)/\tilde{Z}}{\pi(A)/Z} = \ln \frac{Z}{\tilde{Z}} \leq \ln \frac{1}{1-\epsilon} = \ln\Big(1 + \frac{\epsilon}{1-\epsilon}\Big) \leq \frac{\epsilon}{1-\epsilon}\,.$$

Here the first equality holds because $\tilde{\pi}(A) = \pi(A)$ when $\tilde{\pi}(A) > 0$. The first inequality follows from part (i) and the second from the fact that $\ln(1+z) \leq z$ for all $z \geq -1$.

To prove claim (iii), write the total variation distance as

$$\frac{1}{2} \sum_A \Delta(A)\,, \quad \text{where } \Delta(A) := |\pi(A)/Z - \tilde{\pi}(A)/\tilde{Z}|\,.$$

Now observe that $\Delta(A) = \pi(A)/Z$ if $\tilde{\pi}(A) = 0$, and $\Delta(A) = (Z/\tilde{Z} - 1)\pi(A)/Z$ otherwise. We get

$$\begin{aligned}
\sum_A \Delta(A) &= \sum_{A : \tilde{\pi}(A) = 0} \Delta(A) + \sum_{A : \tilde{\pi}(A) > 0} \Delta(A) \\
&= \sum_{A : \tilde{\pi}(A) = 0} \pi(A)/Z + (Z/\tilde{Z} - 1) \sum_{A : \tilde{\pi}(A) > 0} \pi(A)/Z \\
&= 1 - \sum_A \tilde{\pi}(A)/Z + (Z/\tilde{Z} - 1) \sum_A \tilde{\pi}(A)/Z \\
&= 2(1 - \tilde{Z}/Z) \\
&\leq 2\epsilon\,,
\end{aligned}$$

where the last inequality follows from claim (i). This completes the proof.

Table 3: Recent Bayesian DAG samplers (sorted by publication date)

Name & reference	Approach	Notes
Gibby this paper	MCMC in the DAG space	New algorithmic techniques to expedite both basic moves and moves requiring sums over parent sets.
BiDAG Kuipers et al. (2022)	MCMC in the space of ordered node partitions	Fast moves between partitions using a lookup table for score sums, assuming a small set of candidate parents.
DAG-GFlowNet Deleu et al. (2022)	Independent sampling using generative flows	Assumes a GPU is available.
DiBS Lorch et al. (2021)	Gradient-based sampling of DAGs using latent variables and a continuous relaxation of the acyclicity constraint	Computes the posterior distribution of DAGs and supports the BGe score.
BayesDAG Annadani et al. (2023)	Stochastic gradient MCMC and variational inference in the joint structure–parameter space	Assumes continuous data and an additive-noise model. Does not apply to categorical data.
BCD Nets Cundy et al. (2021)	Variational inference with a limited class of approximating distributions	Assumes continuous data and a linear model.
Gadget Viinikka et al. (2020)	MCMC in the space of ordered node partitions	A more scalable implementation of *BiDAG*. Poor accuracy reported (Deleu et al., 2022). Code apparently not available at the moment.
Layering-MCMC Viinikka and Koivisto (2020)	MCMC in the space of layers of ordered node partitions	The implementation not made to be scalable to networks on dozens of nodes. Suffers from similar mixing problems than partition MCMC that does not use parent set resampling moves. For ZOO MAD values around 0.05 reported (Viinikka and Koivisto, 2020, Fig. 4).
Gibbs Goudie and Mukherjee (2016)	MCMC in the DAG space	Slow for larger max-indegree. Does not allow for score pruning using the technique of this paper.
BEANDisco Niinimäki et al. (2016)	MCMC and annealed importance sampling in the space of bucket orderings of nodes	Non-uniform structure prior over equivalent DAGs.

A.4 EXACT SAMPLING AND SUMMING OVER DAGS

The exact arc posterior probabilities can be computed in time $O(3^n n)$ by the algorithm of Tian and He (2009). An efficient implementation of this algorithm is given by Pensar et al. (2020), who also extend it to compute, within the same complexity, the ancestor relation probabilities for all pairs of nodes. We used this implementation from the *BIDA* package (released under the MIT license, `https://github.com/jopensar/BIDA`) in the present work for datasets on 20 or fewer variables.

There are also exponential-time algorithms for exact sampling of DAGs from modular distributions (Harviainen and Koivisto, 2024; Talvitie et al., 2020). With t independently sampled DAGs, one can estimate the probability of any structural feature to within an absolute error of $O\big(1/\sqrt{t}\big)$ with high probability, thanks to basic properties of Monte Carlo integration. These sampling algorithms can be particularly useful in (rare) cases where the exact algorithms encounter numerical challenges, such as catastrophic cancellations. We faced such a case with the dataset generated from the CHILD BN and used the implementation of Harviainen and Koivisto (2024) (released in the supplementary material of the article, with no specified license) to estimate the arc posterior probabilities using a sample of 10 000 DAGs.

B BAYESIAN DAG SAMPLERS

Table 3 gives a summary of some recent methods for Bayesian sampling of DAGs. The first three methods are empirically compared in this paper. The rest are included to highlight related methods we did not include in our empirical study for varying reasons: (i) the method or its available implementation is not applicable to categorical data or does not support score

equivalent structure priors (*BayesDAG*, *BCD Nets*, *BEANDisco*), (ii) the method takes the same approach as *Gibby* and, therefore, can be improved by the techniques presented in this paper (*Gibbs*), or (iii) the method takes the same approach as one of the tested methods and the available implementation either is not designed to be scalable or appears to suffer some technical issues (*Layering-MCMC*, *Gadget*).

***Gibby*.** We used *Gibby* in the sampling accuracy studies (Section 5) as follows. We tuned the parameters to match the running time of *BiDAG* and to output the same number of DAGs (except for ANDES, for which we generated 10^4 DAGs). For the small networks, we let *Gibby* take 100 basic moves (i.e., steps) and 2 REV moves per 1 MBR move. For the larger networks, the MBR move proved particularly useful and we let *Gibby* take 50 basic moves and 1 REV move per 1 MBR move. In all runs, we set the accuracy parameter ϵ to 2^{-15} and started from the empty DAG. *Gibby* can take discrete datasets as input and compute the BDeu score for any needed parent set. However, the current implementation does not compute the BGe score directly for continuous data. Instead, it can take as input precomputed local scores in the `.jkl` format, which is the standard output format of the score calculator included in the *GOBNILP* system (Bartlett and Cussens, 2017; Cussens, 2020).

***BiDAG*.** We used *BiDAG* (released under the GPL-2 and GPL-3 licenses) with its default settings, as follows. For the small networks, we aimed at running times ranging from 20 to 40 minutes. To match this range, we let the sampler perform $6 \cdot 10^6$ iterations for ASIA, $7 \cdot 10^6$ iterations for SACHS and ZOO, and $4 \cdot 10^6$ iterations for CHILD. We set the thinning parameter to reduce the sampled DAGs to $1 \cdot 10^5$. For the larger networks we aimed at running times ranging from 10 to 15 hours. Consequently, we let the sampler perform $1 \cdot 10^8$ iterations for HAILFINDER and $4 \cdot 10^7$ iterations for ANDES. The thinning parameter was set to 10^4 for both datasets. Despite the lower number of iterations for ANDES, we used the same thinning factor to ensure that DAG sampling does not dominate the total time requirement (moves between ordered partitions being fast as they do not require computationally expensive DAG sampling).

***DAG-GFlowNet*.** We used *DAG-GFlowNet* (released under the MIT license) with its default settings: We ran the optimization phase with 100 000 iterations and a learning rate of 10^{-5} and then sampled 1 000 DAGs. We were not able to run *DAG-GFlowNet* on datasets over 17 or more variables: the process froze and made no progress even after 12 hours; see Section C for technical information about our computing environment. We tested using a dataset of size 1 000 generated from the CHILD benchmark BN ($n = 20$), the ZOO dataset of size 101 ($n = 17$), and the *Cervical Cancer Behavior Risk* dataset of size 72 ($n = 20$) from the UCI Machine Learning Repository `https://doi.org/10.24432/C5402W`).

***DiBS*.** We used the *DiBS* implementation available at `https://github.com/larslorch/dibs` (released under the MIT license). We used the BGe score, compatible with both *BiDAG* and *Gibby*. To increase the number of sampled DAGs, we set the number of particles to 200. For the experiments on the GAUSSIAN10 dataset, we ran 10 000 gradient-based iterations, which took approximately 20 minutes per run. All other parameters were kept at their default settings.

C COMPUTING ENVIRONMENT

We have implemented *Gibby* in C++ and compiled using *GCC* with the flags `-march=native -O3`. The source code is available at `https://github.com/sums-of-products/gibby`.

Gibby and *BiDAG* were run as single-thread processes on a cluster with AMD EPYC 7452 processors with 32 cores, running at 2 350 MHz, and with 504 GB of RAM.

DAG-GFlowNet was run on a machine with an NVIDIA GeForce RTX 4070 card (5 888 CUDA cores).

DiBS was run on an Apple M3 chip, with 16 GB of RAM.

D DATA GENERATION AND PREPROCESSING

The benchmark BNs we used are from Scutari's Bayesian Network Repository (Scutari, 2022) and licensed under CC BY-SA 3.0.

The datasets we used are from the UCI Machine Learning Repository (Kelly et al., 2023) and licensed under CC BY 4.0.

From the benchmark Bayesian networks, data were generated in a standard way by forward sampling. We only included in our tests one generated dataset of each size.

Table 4: Speed of *Gibby* vs. *GC* on datasets sampled from benchmark Bayesian networks

				Steps per microsecond			Acceptance ratio (%)	
Network	Nodes	Arcs	Max-indeg	*GC*	*Gibby*†	*Gibby*‡	a	b
Dataset size 2 000								
ALARM	37	46	4	3.6	**76**	51	0.35	4.4
HAILFINDER	56	66	4	3.2	**106**	92	0.24	1.3
PATHFINDER	109	195	5	3.0	**31**	30	0.28	1.5
ANDES	223	338	6	1.9	59	**65**	0.038	0.21
PIGS	441	592	2	3.1	617	**628**	0.014	0.14
Dataset size 500								
ALARM	37	46	4	3.3	**58**	46	0.61	3.5
HAILFINDER	56	66	4	3.0	**70**	68	0.58	1.1
PATHFINDER	109	195	5	1.7	**22**	21	0.40	1.2
ANDES	223	338	6	1.5	31	**36**	0.075	0.15
PIGS	441	592	2	2.2	**1068**	984	0.0083	0.064

†Constant-time acyclicity checks by maintaining the ancestor relation ‡Acyclicity checks by path-finding algorithms

In addition, we generated a dataset with continuous variables, the GAUSSIAN10 dataset. It consists of 100 data points sampled from a 10-node Gaussian network, which we generated using the corresponding function implemented in *DiBS* Lorch et al. (2021). The underlying graph is sampled from an Erdős–Rényi DAG model. Specifically, a DAG is constructed by first sampling a topological ordering of the nodes and then adding edges independently with probability p from earlier to later nodes in the order, ensuring acyclicity. We set $p = 0.5$. Given the DAG, the weights of the linear Gaussian model are generated as follows: for each edge ij in the DAG, a weight w_{ij} is sampled from $\mathcal{N}(0, 1)$ and then shifted by $0.5\,\mathrm{sign}(w_{ij})$ to increase the signal strength.

We acknowledge that independently generated replicas of the datasets could yield slightly different numerical results. The generated datasets are available at `https://github.com/sums-of-products/gibby`.

We did no further preprocessing of the data; there are no missing values; we kept all variables of the benchmark datasets.

E EMPIRICAL RESULTS

This section reports some additional empirical results.

E.1 *GIBBY* VERSUS *GC*

Table 4 shows the speed of *GC* and *Gibby* on datasets of size 2 000 and 500. Both algorithms are faster on the larger datasets when measured by the number of steps taken in a second. This is explained by the smaller acceptance ratio, which is associated with more steps corresponding to rejected proposals and thus less computation required for updating data structures.

However, the largest of the networks, PIGS, shows a different pattern: The tentative acceptance rate b is lower for the smaller dataset, which helps *Gibby*. On the contrary, *GC* becomes somewhat slower on the smaller dataset, which might be explained by the larger acceptance rate a (PIGS being an exception, however).

E.2 EFFICIENCY OF SCORE PRUNING

Table 5 shows the efficiency of score pruning on datasets of size 2 000 and 500. We observe that the data size has a clear effect on pruning efficiency, the larger size either lowering or increasing the fraction of kept parent sets. For the HAILFINDER and PIGS datasets, the savings are lower on the smaller size (500), while for the other cases pruning is more efficient for the larger size (2 000).

Table 5: Efficiency of pruning on datasets sampled from benchmark Bayesian networks

				$\epsilon = 2^{-15}$			$\epsilon = 2^{-10}$		
Network	Nodes	Max-indeg	$K^\dagger$	Kept (%)	$T_\mathrm{b}{}^*$	T_c	Kept (%)	T_b	T_c
				Dataset size 2000					
ALARM	37	4	36	4.0	9.4	17	3.0	8.7	17
HAILFINDER	56	4	55	0.14	9.3	151	0.13	9.1	150
PATHFINDER	109	5	64	14	9930	24200	9.3	8540	24900
ANDES	223	6	64	0.055	5470	–	0.017	1980	–
PIGS	441	2	64	35	13	13	34	13	13
PIGS	441	4‡	64	0.35	211	6810	0.34	206	6980
				Dataset size 500					
ALARM	37	4	36	8.8	7.6	11	4.9	6.7	11
HAILFINDER	56	4	55	0.067	5.1	98	0.064	5.0	98
PATHFINDER	109	5	64	28	9004	16627	19	7829	14563
ANDES	223	6	64	0.46	21711	–	0.10	7154	–
PIGS	441	2	64	14	8.7	8.7	13	8.7	8.7
PIGS	441	4‡	64	0.076	73	4463	0.070	65	4380

†No. candidate parents per node *No. seconds by bottom-up (T_b) and complete pruning (T_c) ‡Larger than in the generating network

Table 6: The values of the max-indegree parameter used in the experiments

Name	Nodes	Max-indeg	Explanation
			Small networks
ASIA	8	7	The largest possible
SACHS	11	10	The largest possible
CHILD	20	9	The total number of possible parent sets to not exceed 10^7
ZOO	17	13	The maximum number of parents *BiDAG* can handle (the `hardlimit` parameter)
			Large networks
HAILFINDER	56	4	The max-indegree of the data-generating DAG
ANDES	223	6	The max-indegree of the data-generating DAG

E.3 ACCURACY OF POSTERIOR SAMPLING

For the experiments, we had to set the max-indegree parameter, which gives an upper bound for the number of parents of any node. When viewed as part of the structure prior, the value of the parameter affects the (true) posterior distribution. Since we were able to deal with the exact posterior distributions for the small networks ($n \leq 20$), we set the max-indegree as large a value as possible. We summarize these choices in Table 6.

Figures 3–5 compare the three DAG samplers on the datasets with 20 or fewer variables. When all the five independent runs of a sampler converge to about the same score range, then the estimates for the arc posterior probabilities also concentrate around the exact probability. The shown traceplots and correlation plots partly explain the MAD curves of *Gibby* and *BiDAG* shown in Fig. 1. We managed to run *DAG-GFlowNet* successfully for ASIA and SACHS, for which it used about 30 minutes in the learning phase, after which it relatively quickly produced 1 000 independent DAG samples. We observe that the best runs of *DAG-GFlowNet* give good estimates for ASIA but not for SACHS.

Figure 7 shows the correlation of arc posterior probability estimates of different runs of the samplers on dataset generated from the larger benchmark BNs. The results suggest that *Gibby* is able to reliably produce accurate estimates.

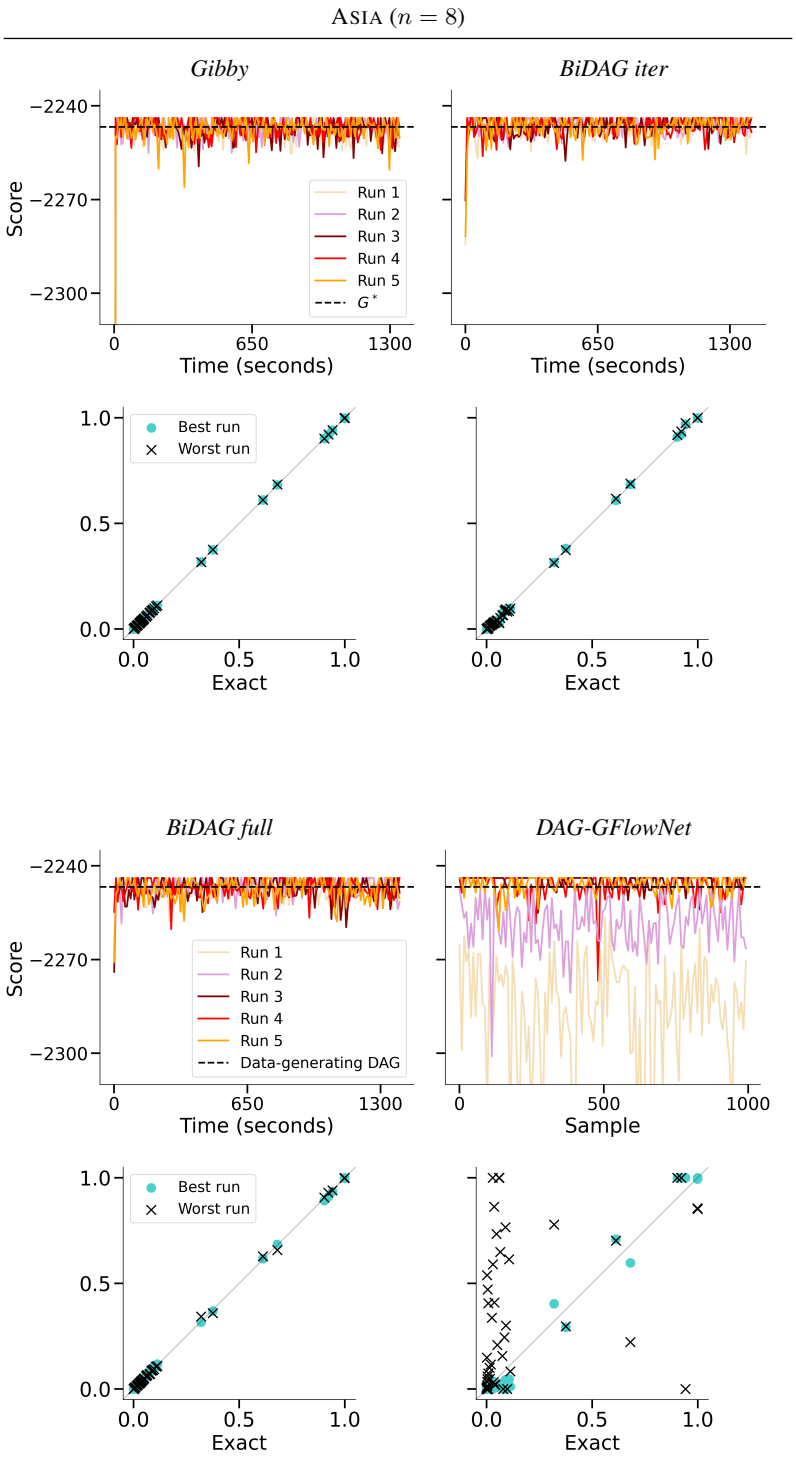

Figure 3: Performance of DAG samplers on an ASIA dataset of size 1 000. *Top:* The log posterior probability of the sampled DAG in five independent runs. The posterior probability of the data-generating DAG G^* is marked by a horizontal black line. *Bottom:* The estimated arc posterior probabilities at the end of the runs, against the exact values. For each node pair, the best and the worst estimate over the five runs are shown.

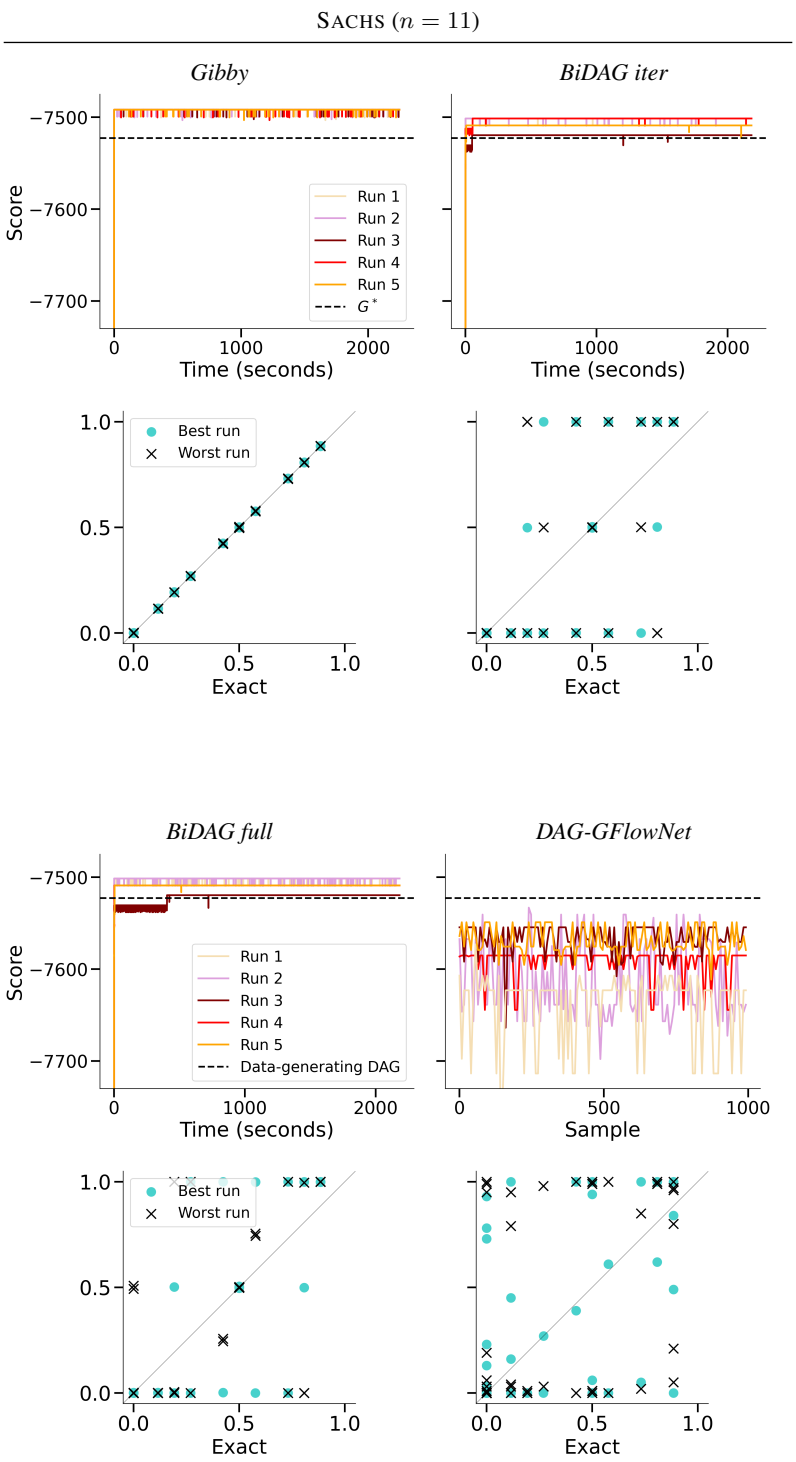

Figure 4: Performance of DAG samplers on an SACHS dataset of size 1 000. *Top:* The log posterior probability of the sampled DAG in five independent runs. The posterior probability of the data-generating DAG G^* is marked by a horizontal black line. *Bottom:* The estimated arc posterior probabilities at the end of the runs, against the exact values. For each node pair, the best and the worst estimate over the five runs are shown.

E.4 COMPARISONS ON CONTINUOUS DATA

Finally, we compare the performance of *Gibby*, *BiDAG* (Suter et al., 2023), and *DiBS* (Lorch et al., 2021) on continuous data, as *DiBS* cannot handle discrete data and the BDeu score.

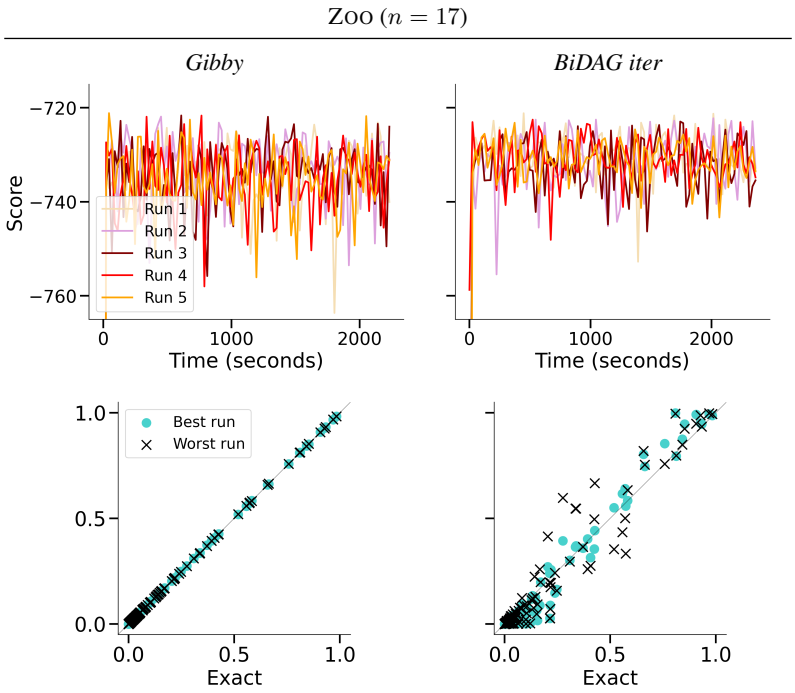

Figure 5: Performance of DAG samplers on the ZOO dataset (size 101). *Top:* The log posterior probability of the sampled DAG in five independent runs. *Bottom:* The estimated arc posterior probabilities at the end of the runs, against the exact values. For each node pair, the best and the worst estimate over the five runs are shown.

DiBS is a variational method for learning Bayesian network structures. It represents candidate graphs using continuous latent variables, referred to as particles, with each particle inducing a sampled graph. The algorithm uses gradient-based optimization, and the objective includes a differentiable penalty term to discourage cyclicity (Yu et al., 2019). The standard *DiBS* implementation computes the posterior distribution by assigning uniform weights to each sampled DAG. In the comparison, we further included *DiBS+*, which uses a score-based weighting of the DAGs.

We ran the experiments on the GAUSSIAN10 dataset, that contains 100 data points sampled from a Gaussian network of 10 nodes; see Section D for more details.

We used the BGe score, a standard choice for linear Gaussian models, along with the uniform prior, which is easily implemented across all algorithms. Each algorithm was run five times, with each run lasting 20 minutes. The number of sampled DAGs for *BiDAG* and *Gibby* was set to 100 000. To have a more meaningful comparison, we increased the default number of particles *DiBS* to 200.

In Figure 8, we report the MAD over time for each single run. The reference distribution has been computed using *BIDA* (Viinikka et al., 2020). *Gibby* outperforms the other methods. Notably, *BiDAG iter* is not able to efficiently extend the state space to all high-scoring DAGs, resulting in a significant posterior mass loss. *DiBS* and *DiBS+* fail to produce a good approximation of the posterior distribution, with MAD not significantly decreasing over time. We chose not to conduct further experiments on continuous data given the observed poor performance of *DiBS*, with respect to our metrics, already on the easy 10-node case.

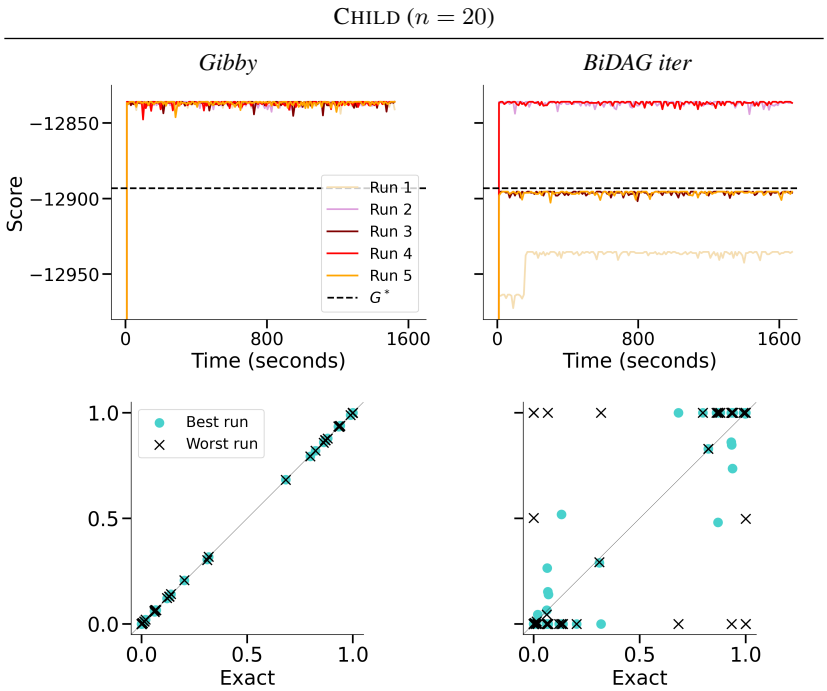

Figure 6: Performance of DAG samplers on a CHILD dataset of size 1 000. *Top:* The log posterior probability of the sampled DAG in five independent runs. The posterior probability of the data-generating DAG G^* is marked by a horizontal black line. *Bottom:* The estimated arc posterior probabilities at the end of the runs, against the exact values. For each node pair, the best and the worst estimate over the five runs are shown.

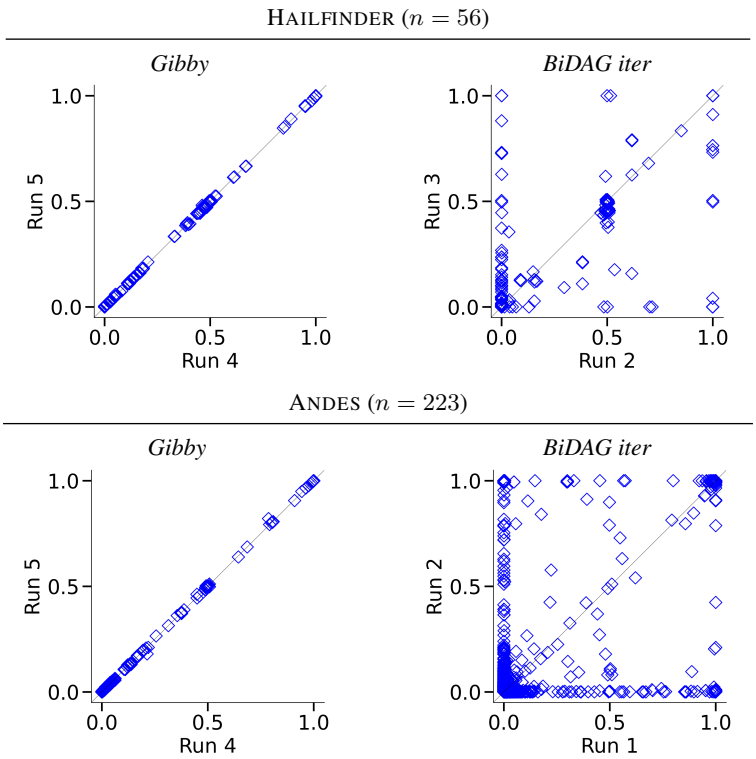

Figure 7: Performance of DAG samplers on HAILFINDER and ANDES datasets of size 1 000. For each sampler and dataset, shown are the estimated arc posterior probabilities for the "worst" pair out of five independent runs. The shown pair was selected to maximize the absolute difference of the estimates over all possible arcs.

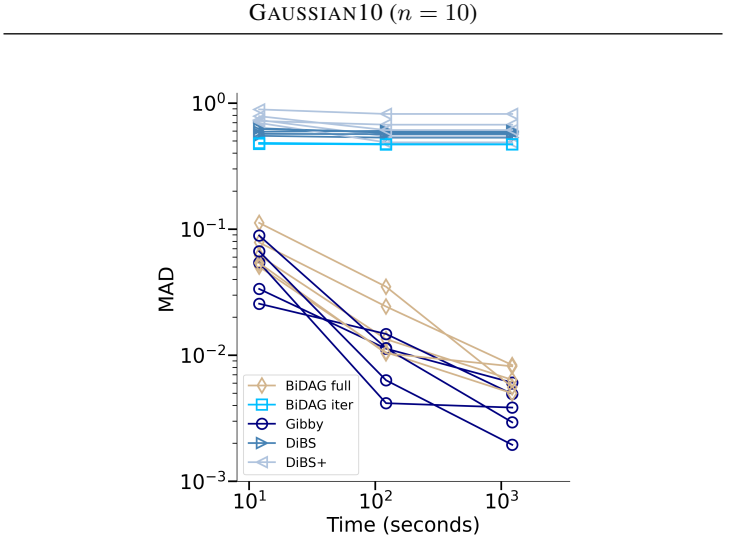

Figure 8: Performance on the GAUSSIAN10 dataset (100 data points). Shown is the MAD of five independent runs as a function of running time.