Layering-MCMC for Structure Learning in Bayesian Networks

Jussi Viinikka
Department of Computer Science
University of Helsinki
jussi.viinikka@helsinki.fi

Mikko Koivisto
Department of Computer Science
University of Helsinki
mikko.koivisto@helsinki.fi

Abstract

Bayesian inference of the Bayesian network structure requires averaging over all possible directed acyclic graphs, DAGs, each weighted by its posterior probability. For approximate averaging, the most popular method has been Markov chain Monte Carlo, MCMC. It was recently shown that collapsing the sampling space from DAGs to suitably defined ordered partitions of the nodes substantially expedites the chain's convergence; this partition-MCMC is similar to order-MCMC on node orderings, but it avoids biasing the sampling distribution. Here, we further collapse the state space by merging some number of adjacent members of a partition into layers. This renders the computation of the (unnormalized) posterior probability of a state, called layering, more involved, for which task we give an efficient dynamic programming algorithm. Our empirical studies suggest that the resulting layering-MCMC is superior to partition-MCMC in terms of mixing time and estimation accuracy.

1 INTRODUCTION

The Bayesian paradigm for statistical inference calls for marginalizing the posterior distribution over all unknowns (variables, parameters) that are of no direct interest. For Bayesian inference in graphical models with unknown structure, Madigan and York (1995) implemented the Bayesian approach using the Markov chain Monte Carlo (MCMC) method. Their *structure-MCMC* simulates a Markov chain on the space of directed acyclic graphs (DAGs) by stochastically adding, removing, and reversing arcs, yielding a sample of DAGs drawn approximately from the posterior distribution. While non-trivial exact algorithms for model averaging and sampling have been developed later (Tian and He, 2009; Talvitie et al., 2019), it is the MCMC method that holds a promise of applicability to graphs on dozens or hundreds of nodes.

Subsequent works have improved upon structure-MCMC using various techniques. Friedman and Koller (2003) proposed *order-MCMC* to simulate a Markov chain on node orderings, each of which covers a large set of DAGs. The smaller state space and smoother posterior landscape of order-MCMC improve the mixing of the chain, which appeared sufficient to compensate a larger computation time per simulation step. Niinimäki et al. (2016) collapsed the state space further by sampling partial orders, with favorable effect on mixing properties. Unfortunately, these ordering based methods introduce a bias by favoring DAGs that are compatible with a larger number of node orderings. Ellis and Wong (2008) proposed a heuristic to correct the bias; Niinimäki et al. (2016) corrected the bias through respective importance sampling weights, i.e., the numbers of topological sorts of the DAGs generated from the sampled partial orders. Neither of these correction methods can escape the fact that the biased sampling distribution can be relatively far from the original target distribution.

Another line of works to improve upon structure-MCMC insist on unbiased sampling. Grzegorczyk and Husmeier (2008) proposed a stronger arc reversal move that not only reverses an arc but at the same time reassigns the parents of its both ends, significantly expediting the mixing of the chain. Kuipers and Moffa (2017) made a further improvement by collapsing the state space to ordered partitions of the nodes. In their *partition-MCMC*, like in order-MCMC, the better mixing of the Markov chain outweighs the increased computational cost per simulation step; the mixing appeared comparable to that of order-MCMC, yet avoiding the sampling bias.

In this paper, we investigate a way to further collapse the state space of partition-MCMC by grouping adjacent members of a partition into what we will call layers.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

This idea stems from two observations: First, on typical benchmark datasets, partition-MCMC tends to sample partitions that consist of a large number of small node subsets, often containing just one or two nodes; see Section 4.1 for our empirical findings. Consequently, the effective sampling space is relatively large, comparable to that of order-MCMC. In contrast, a layering consists of a relatively small number of larger layers, each layering covering a large number of partitions. This renders the sampling space smaller and smoother.

Second, each simulation step in partition-MCMC requires summing over, in essence, all possible parent sets for each node, typically millions of sets when there are some dozens of nodes; this computation is similar to what is required in order-MCMC. Given that the step is in any case computationally demanding, in layering-MCMC we can afford the additional work needed for computing the (unnormalized) posterior probability of a given layering, namely, for summing over all partitions (and thereby DAGs) compatible with the layering. Indeed, we show that, following the recent approach of Talvitie et al. (2019), we can compute the sum in time that is moderately exponential in the largest layer size; by controlling the maximum layer size, we can trade the computational complexity of a simulation step for the smoothness of the posterior landscape.

This paper focuses on the key ideas of layering-MCMC. The main question we aim to answer is the following:

> Does collapsing the sampling space from partitions to layerings significantly enhance the mixing properties of the Markov chain and the accuracy of the resulting MCMC estimator for arc posterior probabilities?

Here we are bound to set the maximum layer size so that computing the posterior probability of a layering does not become a computational bottleneck. To answer the above question, we will compare layering-MCMC to partition-MCMC running their basic instantiations on data sets of moderate dimensions, for which the computation of the exact arc posterior probabilities is feasible (Tian and He, 2009; Talvitie et al., 2019).

If the answer is affirmative, one can expect the collapsed state space to preserve its advantage when one enhances the basic instantiations by sophisticated MCMC techniques, such as Metropolis-coupling of several parallel, "heated" chains; supporting evidence for this viewpoint is provided by the analogous comparison of order-MCMC and partial-order-MCMC (Niinimäki et al., 2016). Such additional techniques are needed and available when one wishes to successfully infer DAGs from data sets of larger dimensions. However, implementing such enhancements is beyond the present work.

2 PRELIMINARIES

We begin by introducing the necessary concepts and notation related to inferring BNs from data; for more comprehensive background on Bayesian networks, we refer to Koller and Friedman (2009). We also review the essential ingredients of the partition-MCMC method of Kuipers and Moffa (2017).

2.1 BAYESIAN NETWORKS

Let V be a set of n elements. With each $v \in V$ associate a random variable X_v. If $S \subseteq V$, write X_S for the tuple $(X_v)_{v \in S}$. A BN over the variables is a pair (G, θ), where G is a DAG on V and θ parameterizes a joint probability distribution of X_V that factorizes along the DAG: $p(X_V|G, \theta) = \prod_{v \in V} p(X_v|X_{G_v}, G, \theta)$. Here and henceforth G_v denotes the set of parents of v in G. We also let p stand for a generic name for a probability distribution; the referred random variables and events will be clear from the context. The notation anticipates the Bayesian approach, in which the uknown DAG G and its parameters θ will be treated as random variables.

In principle, the parent set G_v of node v could be any subset of $V \setminus \{v\}$. However, we will be interested in scenarios where each node v is associated with some potentially much smaller family of *possible parent sets*, denoted by $\mathcal{G}_v$. For instance, $\mathcal{G}_v$ might consist of all $G_v \subseteq V \setminus \{v\}$ whose size is at most some fixed constant K, the *max-indegree* of the allowed DAGs, or it could consist of all subsets of some fixed set of *candidate parents* $C_v \subseteq V \setminus \{v\}$. We will denote by $\mathcal{G}$ the resulting set of all DAGs G satisfying $G_v \in \mathcal{G}_v$ for all $v \in V$.

2.2 BAYESIAN INFERENCE OF THE NETWORK STRUCTURE

We will consider Bayesian inference of the structure G from a given *dataset* $\mathbf{X}$. We will assume that $\mathbf{X}$ consists of N datapoints $X^1, X^2, \ldots, X^N$, each of which is viewed as an independent draw from the distribution associated with the DAG G. Consequently, the likelihood of the BN (G, θ) is obtained as $\prod_{s=1}^{N} p(X^s|G, \theta)$. The *marginal likelihood* $p(\mathbf{X}|G)$ is obtained by integrating the likelihood over a *parameter prior* $p(\theta|G)$. For any *structure prior* $p(G)$, the structure posterior is obtained via the Bayes rule as $p(G|\mathbf{X}) = p(G)p(\mathbf{X}|G)/p(\mathbf{X})$. We will denote the posterior of G also by $\pi(G)$. The posterior $\pi(G)$ enables inference on any function of the structure. For instance, the posterior probability that node u is a parent of node v is obtained by simply marginalizing

the posterior, i.e., summing up the probabilities $\pi(G)$ of all DAGs G in which $u \in G_v$.

In our experiments we adopt some standard choices for the priors so that the marginal likelihood $p(\mathbf{X}|G)$ satisfies so-called score equivalence, i.e., is constant over Markov equivalent DAGs (Koller and Friedman, 2009, Sect. 18.3.7). Specifically, we assume that all variables are either categorical or continuous, and use the BDe score in the former and the BGe score in the latter case (Geiger and Heckerman, 2002; Kuipers et al., 2014). Furthermore, we let the structure prior $p(G)$ be uniform (truncated to $\mathcal{G}$).

These choices imply that the posterior $\pi(G)$ is *modular*:

$$\pi(G) := \prod_{v \in V} \pi_v(G_v)\,,$$

where $\pi_v(G_v)$ can be efficently computed up to a constant factor that is independent of G_v. In fact, the computational methods we will describe only rely on this property of the posterior.

2.3 PARTITION-MCMC

The partition-MCMC method (Kuipers and Moffa, 2017) stems from the fact that every DAG admits a unique ordered partition of its node set, obtained be iteratively extracting the root nodes of the DAG; a node is a *root node* if it has no parents. For a graph G and a subset of its nodes S, denote by $G - S$ the graph obtained by removing the nodes in S and all edges that have an end in S.

Definition 1. The *root-partition* of a DAG G is the sequence $R_1 R_2 \cdots R_k$, where R_1 is the set of root nodes of G and, if $k > 1$, the remainder $R_2 R_3 \cdots R_k$ is the root-partition of $G - R_1$.

Denote by $\mathcal{R}$ the set of all ordered set partitions of the node set V. While the root-partition of any DAG is unique, every member of $\mathcal{R}$ is the root-partition of one or multiple DAGs on V. With every $R \in \mathcal{R}$ we associate its posterior probability $\pi(R)$ obtained by marginalizing the posterior of DAGs:

$$\pi(R) = \sum_{G \in \mathcal{G}(R)} \pi(G)\,,$$

where

$$\mathcal{G}(R) := \{G \in \mathcal{G} : R \text{ is the root-partition of } G\}\,.$$

Partition-MCMC runs the Metropolis–Hastings algorithm to simulate a Markov chain on $\mathcal{R}$ with the posterior $\pi(R)$ as its stationary distribution. When the chain is in state R, the next state is generated by first drawing a proposal state R' from a proposal distribution $q(R'|R)$ and then accepting the proposal with probability

$$\min\left\{1, \frac{\pi(R')}{\pi(R)} \frac{q(R|R')}{q(R'|R)}\right\}, \tag{1}$$

in which case the next state is R', and otherwise setting the next state to R. Thus the proposal distribution can be viewed as generating possible *moves* in the state space.

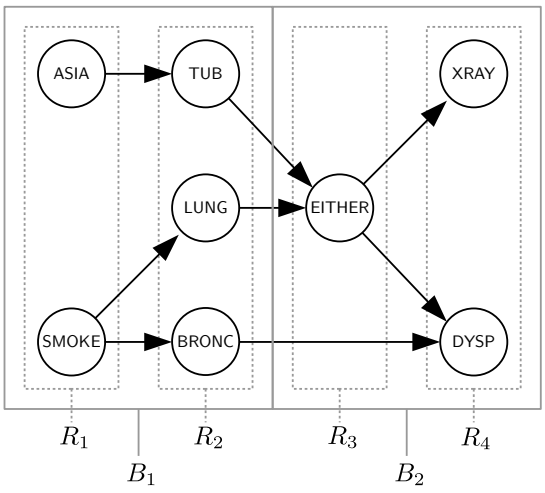

Figure 1: The root-partition R and the 5-layering B of the ASIA network. See Examples 1 and 2.

Kuipers and Moffa (2017) consider various types of moves in the partition space: splitting a part into two adjacent parts; joining two adjacent parts into a single part; moving a single node to another part or to form a new part of size one; swapping two nodes from different parts. In addition, they employ the edge reversal move of Grzegorczyk and Husmeier (2008) by simply first generating a DAG conditionally on the current partition, making the edge reversal move in the DAG space, and mapping the obtained DAG to its root-partition.

Example 1. The root-partition of the eight-node ASIA benchmark network (Lauritzen and Spiegelhalter, 1988) consists of four parts (Fig. 1). Observe that there are no arcs within any part and that every node, unless it is in the first part, takes a parent from the previous part.

2.4 COMPUTING THE PARTITION POSTERIOR

Each simulation step in partition-MCMC requires the computation of the posterior $\pi(R)$ of a given partition $R = R_1 R_2 \cdots R_k$, up to a normalizing constant (which cancels when computing the ratio (1)). Instead of directly summing over all DAGs compatible with R, a computationally more efficient way is enabled by the fact

that conditionally on the partition, the nodes' parent sets are mutually independent. Furthermore, a $G_v \in \mathcal{G}_v$ is a valid parent set of a node $v \in R_i$ exactly when G_v only contains nodes from members R_j for $j < i$ and at least one parent from R_{i-1}; the obvious exception is when $i = 1$, in which case G_v must be empty.

For a more precise formulation of this crucial property, for any node $v \in V$ and node subsets $T \subseteq U \subseteq V$, denote the possible parent sets of v relative to (U, T) by

$$\mathcal{G}_v(U,T) := \{G_v \in \mathcal{G}_v : G_v \subseteq U, G_v \cap T \neq \emptyset\}$$

if U is not empty and $\mathcal{G}_v(\emptyset,\emptyset) := \mathcal{G}_v \cap \{\emptyset\}$ otherwise, and the corresponding sum of posterior weights by

$$\hat{\pi}_v(U,T) := \sum_{G_v \in \mathcal{G}_v(U,T)} \pi_v(G_v)\,.$$

Now, by letting

$$f(U,T,S) := \prod_{v \in S} \hat{\pi}_v(U,T)\,,$$

we have that $\pi(R)$ factorizes into a part-wise product:

$$\pi(R) = \prod_{i=1}^{k} f(R_{1:i-1}, R_{i-1}, R_i)\,; \tag{2}$$

here and henceforth, $R_{-1}, R_{1:-1} := \emptyset$. In this product each node $v \in R_i$ contributes a term $\hat{\pi}_v(U,T)$, where U consists of all nodes in R_j for $j < i$ and T equals R_{i-1}.

The factorization reduces the computations to the computation of the required sum $\hat{\pi}_v(U,T)$ for each v. A straightforward way to compute the sum is by enumerating all $G_v \in \mathcal{G}_v(U,T)$, which, in turn, can be implemented by iterating through all $G_v \in \mathcal{G}_v$ and checking for each G_v whether it satisfies the two constraints, $G_v \subseteq U$ and $G_v \cap T \neq \emptyset$. This algorithm is implemented in Kuipers and Moffa (2017).

3 LAYERING-MCMC

The key feature in layering-MCMC is that its state space is smaller than in partition-MCMC (Kuipers and Moffa, 2017). We begin by introducing the notion of M-layering, where M is a user parameter that controls the size of the state space. Then we formulate a Metropolis–Hastings algorithm for simulating a Markov chain on M-layerings, following the corresponding algorithm of partition-MCMC. While it is relatively straightforward to compute the unnormalized posterior probability of a root-partition (see Section 2.4), for M-layerings the respective task is more involved—we will give a dynamic programming algorithm whose time complexity is exponential only in the parameter M and therefore not a computational bottleneck when M is set small enough.

3.1 LAYERINGS

Consider an ordered set partition $R = R_1 R_2 \cdots R_k \in \mathcal{R}$ of the node set V, and let M be a natural number less than or equal to $|V|$. The M-layering of R, defined formally below, is a unique grouping of adjacent members of R into disjoint layers, each containing at most M nodes and being as large as possible; an exception is when a layer corresponds to a single member of R whose cardinality exceeds M.

Definition 2. The *M-layering* of $R_1 R_2 \cdots R_k$ is the sequence $B_1 B_2 \cdots B_\ell$ where

$$B_1 = \begin{cases} R_1 & \text{if } |R_1| > M\,, \\ R_{1:i} & \text{else, for the largest } i \text{ s.t. } |R_{1:i}| \leq M\,; \end{cases}$$

and, if $i < k$, the remainder $B_2 B_3 \cdots B_\ell$ is the M-layering of $R_{i+1} R_{i+2} \cdots R_k$.

Observe that the 1-layering of a root-partition of a DAG is the root-partition itself, and in this sense layerings amount to a generalization of root-partitions.

Example 2. The 5-layering of the eight-node ASIA benchmark network consists of two layers (Fig. 1).

The partitions in $\mathcal{R}$ that are compatible with a given M-layering B form the subset

$$\mathcal{R}(B) := \{R \in \mathcal{R} : B \text{ is the } M\text{-layering of } R\}\,.$$

The posterior probability of B is thus obtained as

$$\pi(B) := \sum_{R \in \mathcal{R}(B)} \pi(R)\,.$$

We denote by $\mathcal{B}_M$ the set of all M-layerings on V, i.e.,

$$\mathcal{B}_M := \{B : B \text{ is the } M\text{-layering of } R \text{ for some } R \in \mathcal{R}\}.$$

The definitions directly imply the following characterization of $\mathcal{B}_M$:

Proposition 1. *An ordered set partition of V is an M-layering on V if and only if the total size of any two adjacent parts exceeds M.*

3.2 MARKOV CHAIN ON LAYERINGS

Our algorithm for simulating a Markov chain on $\mathcal{B}_M$ is analogous to that in partition-MCMC. In particular, the chain moves from the current state B to a new state B', generated from a proposal distribution $q(B'|B)$, with probability

$$\min\left\{1, \frac{\pi(B')}{\pi(B)} \frac{q(B|B')}{q(B'|B)}\right\}. \tag{3}$$

At first glance, one might also consider directly adopting the moves in the partition space. However, the additional constraints in $\mathcal{B}_M$ (cf. Proposition 1) invalidate some moves; for example, splitting a layer of size at most M would always produce an element of $\mathcal{R}$ that does not belong to $\mathcal{B}_M$.

We have implemented three types of moves. In the description below, all draws are uniformly at random over the available choices unless told otherwise.

Relocate Draw a layer and an integer s between 1 and the layer's size. Draw s nodes from the layer and insert them into another layer or in between adjacent two layers (to form a new layer).

Swap With probability one half, draw two adjacent layers; otherwise draw two non-adjacent layers. Draw one node from each layer and swap their locations.

Re-partition First draw a partition $R \in \mathcal{R}(B)$ proportionally to $\pi(R)$. Then make a move in the partition space (split, join, or swap); this results in a partition R'. Finally map R' to the induced layering $B' \in \mathcal{B}_M$, which is accepted as the new state.

For the relocate and swap moves, it is straightforward to calculate the proposal probabilities $q(B'|B)$ and $q(B|B')$, which we need for the evaluation of the ratio (3); we omit the calculation here. For the re-partition move, these terms are not needed, given that the proposed partition R' is accepted according to the respective ratio of posterior and proposal probabilities (1)—the argument is, in essence, given by Kuipers and Moffa (2017, Sect. 5) and not repeated here.

Since we use the Metropolis–Hastings algorithm, the chain is reversible and thus converges to $\pi(B)$, provided that the chain is irreducible and aperiodic. It is easy to see that our chain is irreducible, since from any layering one can move to the single-layer layering by repeated relocate moves, and then back to any other layering by another series of relocate moves. To make the chain aperiodic, we simply let the chain stay in the same state with some small probability.

3.3 COMPUTING THE LAYERING POSTERIOR

Let B be an M-layering on V. By the definition of $\pi(B)$ and the factorization (2), we have

$$\pi(B) = \sum_{R \in \mathcal{R}(B)} \prod_{i=1}^{k(R)} f(R_{1:i-1}, R_{i-1}, R_i)\,.$$

To compute $\pi(B)$ by dynamic programming, we next define a function g_j through a recurrence for all $j = 0, 1, \ldots, \ell$ such that $\pi(B)$ can be read from g_0. Let us make the notational convention that $B_{-1} = B_0 = \emptyset$. Let $T \subseteq D \subseteq B_j$ and denote $U := B_{1:j-1} \cup D$. We next define the value $g_j(U, T)$.

Suppose first that $D = B_j$, i.e., $U = B_{1:j}$. Now, if $j = \ell$, define $g_j(U, T) := 1$; otherwise, if $|B_{j+1}| > M$, define

$$g_j(U, T) := f(U, T, B_{j+1})\, g_{j+1}(B_{1:j+1}, B_{j+1})\,;$$

else define

$$g_j(U, T) := \sum_{\substack{\emptyset \subset S \subseteq B_{j+1} \\ j=0 \text{ or } |S| > M - |B_j|}} f(U, T, S)\, g_{j+1}(U \cup S, S)\,.$$

Suppose then that $U \neq B_{1:j}$, i.e., $D \subset B_j$. Then define

$$g_j(U, T) := \sum_{\emptyset \subset S \subseteq B_j \setminus U} f(U, T, S)\, g_j(U \cup S, S)\,.$$

The proof of the following result is mechanical and, for the sake of exposition, given in the Supplement.

Lemma 2. *We have* $g_0(\emptyset, \emptyset) = \pi(B)$.

Our dynamic programming algorithm computes the values $g_j(U, T)$ using the above recurrences in decreasing order by j and $|U|$. The sets $T \subseteq U \setminus B_j$ can be traversed in arbitrary order, while, for the ease of computation, it is advisable to sum over the relevant sets S in increasing order by size, as discussed in the next paragraph.

The complexity of the algorithm is clearly dominated by the complexity of the steps that require summing over the sets S. In the first summation, U is fixed, while T and S range over subsets of B_j and B_{j+1}, respectively, whose number in total is at most $2^{|B_j|} 2^{|B_{j+1}|} \leq 4^M$. (Observe that if $|B_j| > M$, then $g_j(U, T)$ is only needed for $T = B_j$.) In the second summation, (U, T, S) runs over at most $4^{|B_j|} \leq 4^M$ choices. Since we clearly have $\ell = O(n/M)$, we need $O(4^M n/M)$ arithmetic operations, provided that we can access the values $f(U, T, S)$ using a constant number of operations. To this end, we visit the sets S in increasing order by size and compute $f(U, T, S)$ by multiplying an already computed value $f(U, T, S \setminus \{v\})$ by $\hat{\pi}_v(U, T)$ for some $v \in S$. One way to organize these computations are given in the pseudo code in Fig. 3.

It remains to analyze the complexity of precomputing the values $\hat{\pi}_v(U, T)$ for every node v. For convenience, assume first that T intersects B_j and $v \in B_j$. A straightforward, but slow, approach would be to compute the values separately for all possible pairs (U, T), whose number

scales in the worst case as 3^M. To expedite the computations, we however compute the values simultaneously for all (U, T), as follows. For all $C \subseteq B_j$ define

$$\tau_v(C) := \sum_{\substack{C' \subseteq B_{1:j-1} \\ C' \cup C \in \mathcal{G}_v}} \pi_v(C' \cup C)\,.$$

We can compute all these values by visiting each possible parent set $G_v \subseteq B_{1:j}$ and adding its contribution, namely $\pi_v(G_v)$, to $\tau_v(G_v \cap B_j)$. This requires $O(|\mathcal{G}_v|)$ additions. Then we compute the zeta transform $\hat{\tau}_v$ of τ_v, defined by

$$\hat{\tau}_v(D) := \sum_{C \subseteq D} \tau_v(C)\,, \quad D \subseteq B_j\,,$$

using $O(3^M)$ additions—or asymptotically faster, using $O(2^M M)$ additions (Kennes, 1992)–and finally make use of the following identity:

Lemma 3. *We have* $\hat{\pi}_v(U, T) = \hat{\tau}_v(D) - \hat{\tau}_v(D \setminus T)$.

Proof. Recall that $U = B_{1:j-1} \cup D$. Write

$$\begin{aligned}
\hat{\tau}_v(D) - \hat{\tau}_v(D \setminus T) &= \sum_{C \subseteq D} \tau_v(C) - \sum_{C \subseteq D \setminus T} \tau_v(C) \\
&= \sum_{\substack{C \subseteq D \\ C \cap T \neq \emptyset}} \sum_{\substack{C' \subseteq U \setminus D \\ C' \cup C \in \mathcal{G}_v}} \pi_v(C' \cup C) \\
&= \sum_{\substack{X \subseteq U \\ X \cap T \neq \emptyset \\ X \in \mathcal{G}_v}} \pi_v(X)\,.
\end{aligned}$$

This equals $\hat{\pi}_v(U, T)$ by the definition of $\mathcal{G}_v(U, T)$. □

Consider then the other case, namely that T intersects B_j but $v \in B_{j+1}$. We see that the construction above is valid also in this case. We also observe that if $|B_j| > M$, then we may assume that $U = T = B_j$ and only need the value $\hat{\pi}_v(B_{1:j}, B_j)$, which can be computed by a straightforward pass through $\mathcal{G}_v$.

The pseudo code in Fig. 2 organizes the computation of the functions $\hat{\tau}_v$.

We arrive at the main result of this subsection:

Proposition 4. *Computing* $\pi(B)$ *for a given* M*-layering* B *requires* $O\big(4^M n/M + \sum_v |\mathcal{G}_v|\big)$ *arithmetic operations.*

3.4 GENERATING PARTITIONS AND DAGS

For the re-partition move of our Markov chain we need an algorithm that generates a partition compatible with the current layering B with a probability that is proportional to the posterior of the partition. Having computed the posterior probability $\pi(B)$ and stored the dynamic programming tables g_j, it is a standard routine to generate a random partition by stochasting backtracking. (A pseudo code is given in the Supplement.)

Having generated a partition, we can further generate a compatible DAG with a probability proportional to the posterior of the DAG. The DAG will be a sample from the posterior conditionally on the current layering.

```
PARENTS-SUMS(B_1 B_2 ··· B_ℓ)
 1  B_{ℓ+1} = ∅
 2  for j = 1 to ℓ
 3      for each v ∈ B_{j+1}        // Preparing for T = B_j
 4          τ̂_v[{v}] = 0                    // Initialize
 5          for each G_v ∈ G_v s.t. G_v ⊆ B_{1:j}
 6              if G_v ∩ B_j ≠ ∅
 7                  add π_v(G_v) to τ̂_v[{v}]
 8      if |B_j| ≤ M
 9          for each v ∈ B_j ∪ B_{j+1}
10              for each C ⊆ B_j
11                  τ_v[C] = 0              // Initialize
12              for each G_v ∈ G_v s.t. G_v ⊆ B_{1:j}
13                  add π_v(G_v) to τ_v[G_v ∩ B_j]
                // Compute τ̂_v
14              for each D ⊆ B_j \ {v}
15                  τ̂_v[D] = 0              // Initialize
16                  for each C ⊆ D
17                      add τ_v[C] to τ̂_v[D]
18  return (τ̂_v)_{v∈V}
```

Figure 2: Pseudo code for computing the functions $\hat{\tau}_v$. Note: if $v \in B_{j+1}$, then the returned value $\hat{\tau}_v[\{v\}]$ equals $\hat{\pi}_v(B_{1:j}, B_j)$; in other cases (i.e., the argument is a proper subset of B_j) one has to invoke Lemma 3.

4 EXPERIMENTS

We have implemented the layering-MCMC method in Python.[1] For tests on partition-MCMC, we used the original implementation by Kuipers and Moffa (2017); for a clean comparison of the different state spaces, we did not enable the edge reversal move. As neither implementation is optimized for speed and the theoretical cost per simulation step is about the same for both algorithms, we run both algorithms the same number of simulations steps, which we set to 60 000. (This is the length of the simulations in Kuipers and Moffa (2017) on data sets of comparable dimensions.)

4.1 TYPICAL ROOT-PARTITIONS

The potential of layering-MCMC crucially relies on the possibility to cover a large number of root-partitions

[1]The source code of our implementation is freely available at `github.com/jussiviinikka/layeringMCMC`.

```
POSTERIOR(B_1 B_2 ··· B_ℓ)
 1  (τ̂_v)_{v∈V} = PARENTS-SUMS(B_1 B_2 ··· B_ℓ)

 2  for j = ℓ downto 0
 3      if |B_j| > M or j == 0                          // Set P
 4          P = {(B_j, B_j)}
 5      else P = {(D,T) : ∅ ⊂ T ⊆ D ⊆ B_j}

 6      for each (D,T) ∈ P in decreasing order by |D|
 7          if D == B_j                                 // j' and D'
 8              j' = j + 1; D' = ∅
 9          else j' = j; D' = D

10          if D ⊂ B_j or j == ℓ or |B_{j+1}| ≤ M // S and A
11              S = {S ⊆ B_{j'} \ D' : ∅ ⊂ S}; A = ∅
12          else S = {B_{j+1}}; A = B_{j+1} \ min B_{j+1}

13          for each v ∈ B_{j'} \ D'                    // Construct p[.]
14              if T == ∅
15                  p[v] = π_v(∅)
16              else if T == B_j                        // Special case
17                      p[v] = τ̂_v[{v}]
18                  else p[v] = τ̂_v[D] − τ̂_v[D\T]

19          f[A] = 1                                    // Construct f[A]
20          for each v ∈ A
21              multiply f[A] by p[v]

22          if S is empty // Happens iff j = ℓ and D = B_j
23              g_j[D,T] = 1
24          else g_j[D,T] = 0
25          for each S ∈ S in increasing order by |S|
26              v = min S
27              f[S] = f[S\{v}] · p[v]
28              if D ≠ B_j or j == 0 or |S| > M − |B_j|
29                  add f[S] · g_{j'}[D' ∪ S, S] to g_j[D,T]
30  return g_0[∅, ∅]
```

Figure 3: Pseudo code for computing the posterior $\pi(B)$ of a given M-layering B. Note: $B_0 = B_{\ell+1} = \emptyset$.

by a M-layering with a relatively small M. To examine whether typical root-partitions permit an effective use of the layering technique, we first collected the size distributions of root-partition parts in benchmark networks (Table 1). In this study, we include all the medium and large networks from the *bnlearn* repository (`www.bnlearn.com/bnrepository`). In addition, we included the ASIA network, which serves as our running toy example. For most of the networks, we see large reductions when comparing the length of the root-partition to the length of the 8-layerings, with WATER and HEPAR II being the only exceptions.

Then we did a similar study for the root-partitions traversed by partition-MCMC on a set of benchmark datasets (Table 2). The datasets are from the UCI machine learning repository (Dua and Graff, 2017), and preprocessed for learning discrete BNs in Malone et al. (2018). We included all datasets with less than 1000 data points and up to 30 variables, to fit within our computational constraints with a maximum indegree of 5. In addition, we included the BOSTON dataset against which partition-MCMC was benchmarked by Kuipers and Moffa (2017), and a dataset of 1000 samples from the ASIA network. We did not set any indegree limit for ASIA as there are only eight nodes.

Arguably, the analysis on the root-partitions visited by a simulated Markov chain is a more direct indicator of the potential of the layering method than the static synthetic networks. Here we see even a greater promise for layering-MCMC: the median sizes of root-partition parts are either 1 or 2, with no exception. Consequently, we see a dramatic reduction from lengths of root-partitions to lengths of 8-layerings.

Table 1: The median length (k) and part size (m) of the root-partition and the length (ℓ) of the 8-layering of benchmark networks

Network	Nodes	Arcs	Max-Indeg.	m	k	ℓ
ASIA	8	8	2	2	4	1
CHILD	20	25	2	5	5	3
INSURANCE	27	52	3	2.5	10	5
WATER	32	66	5	8	4	4
MILDEW	35	46	3	1.5	14	4
ALARM	37	46	4	2	11	5
BARLEY	48	84	4	2.5	12	6
HAILFINDER	56	66	4	1.5	14	6
HEPAR II	70	123	6	8.5	8	8
WIN95PTS	76	112	7	3	9	5

Table 2: The length (k) and part size (m) of the root-partition and the length (ℓ) of the induced 8-layering on benchmark data. Medians of values collected from 60 000 steps of partition-MCMC

Data Set	Nodes	Points	Max-Indeg.	m	k	ℓ
ASIA-1000	8	1000	7	1	6	1
BOSTON	14	506	5	1	10	2
ZOO	17	101	5	1	11	3
VOTING	17	435	5	2	8	3
LYMPH	19	148	5	2	9	3
HEPATITIS	20	155	5	2	10	3
EUCALYPTUS	20	736	5	1	13	3
SPECT	23	267	5	1	16	3
AUTOS	26	205	5	2	16	4
PYRIM	28	74	5	2	14	4
COLIC	28	368	5	2	12	4
FLAG	29	194	5	2	14	4
TRAINS	30	10	5	1	18	4

4.2 MIXING OF MARKOV CHAINS

To study the effect of the maximum layer size M on the performance of layering-MCMC, we ran nine independent chains, with $M \in \{4, 8\}$, on the first six benchmark data sets from Table 2. For comparison, we also ran nine independent chains of partition-MCMC. Figure 4 shows the simulation traces (scores of sampled DAGs) for four selected data sets; see the Supplement for the rest.

For ASIA-1000 and ZOO there is little visible difference between the three algorithms. Both data sets seem relatively easy: all nine chains appear to quickly reach the regions of high posterior probabilities. Here it is worth noting that on ASIA-1000 layering-MCMC with $M = 8$ samples independent DAGs from the exact posterior distribution as the state space only contains one state.

In contrast, for BOSTON and VOTING the simulation traces reveal the difficulty of partition-MCMC to mix between different modes of the distribution. Layering-MCMC with $M = 4$ performs better, but it takes around 10 000 steps before the chains appear to converge, and even then some of the chains on VOTING seem to not reach the high probability regions. Increasing M to 8 has a clear positive effect and the chains appear to converge already after a couple of thousand steps.

4.3 ARC POSTERIOR ESTIMATES

For a less subjective measure of performance we first computed the exact posterior probabilities for edges between each pair of the variables given the computed local scores, with software by Pensar et al. (2020). Then we computed the empirical probability for the same at each iteration step for the three compared MCMC-chains and for each of the 9 independent runs. Finally, we calculated the maximum absolute error between the exact and the empirical probabilities and plot the median of the 9 values thus obtained at each step.

The results are presented in the right-most column in Fig. 4, and clearly show the harshness of the measure. In ASIA and ZOO the curves seem to be able to capture differences between the methods not clearly visible directly from the DAG-scores. Even when the chains seem to have mixed roughly equally well, the analysis on the edge probabilities almost invariantly rank the layering-MCMC with $M = 8$ to have performed the best and partition-MCMC the worst. The only exception happens briefly around 20 000 iterations for BOSTON when partition-MCMC fares better than layering-MCMC with $M = 4$. The overall picture is in line with the theoretical idea of layering-MCMC, despite the more practical differences between the two software implementations.

5 CONCLUDING REMARKS

We proposed the layering-MCMC method for Bayesian structure learning in Bayesian networks. Like the recently discovered partition-MCMC (Kuipers and Moffa, 2017), it allows (approximate) sampling from the *unbiased* posterior distribution of DAGs, thus avoiding the main drawback of some previous sophisticated MCMC methods (Friedman and Koller, 2003; Niinimäki et al., 2016) or bias correction schemes (Ellis and Wong, 2008; He et al., 2016). Because layering-MCMC simulates a Markov chain on layerings, each of which covers multiple ordered partitions, it has the potential to mix faster and thereby yield more efficient estimators of arc posterior probabilities and related quantities, as compared to partition-MCMC.

To realize this potential we gave an algorithm to compute the posterior probability of a given layering suffciently fast, so that the computational complexity is on par with the work required per simulation step in partition-MCMC, or, likewise, in partial-order-MCMC (Niinimäki et al., 2016). More technically, the computational complexity is controlled by the maximum layer size M. The parameter M can be set to roughly match the computational cost due to the large number of potential parent sets, which mainly depends on the number of nodes n and the maximum indegree (if any).

We verified empirically that already a moderate value of the parameter M, say $M = 8$, yields substantial gains in mixing and convergence speed. We can be explain this result by our finding that both in benchmark BNs and in BNs that fit well benchmark data sets, the root-partitions tend to contain a large number small parts, often with just one or two nodes per part; this appears to hold also for larger networks on several dozens of nodes. The present study of layering-MCMC was, however, restricted to moderate-size networks on at most 22 nodes, as motivated by two facts: first, this enables comparison of arc posterior probability estimates to their exact values; second, typical data sets in this category are sufficiently hard for demonstrating clear differences between MCMC schemes.

MCMC methods, of course, aim to scale up to high dimensions that are beyond the reach of exact algorithms. Our results suggest that layering-MCMC holds a promise for reliable inference also in higher dimensions. To fulfill this promise, several enhancements—not included in the present preliminary study—are needed and can be implemented. First, the proposal distributions can be made more efficient in several ways. In particular, it is relatively straightforward to add the so-called new edge reversal move (Grzegorczyk and Husmeier, 2008)

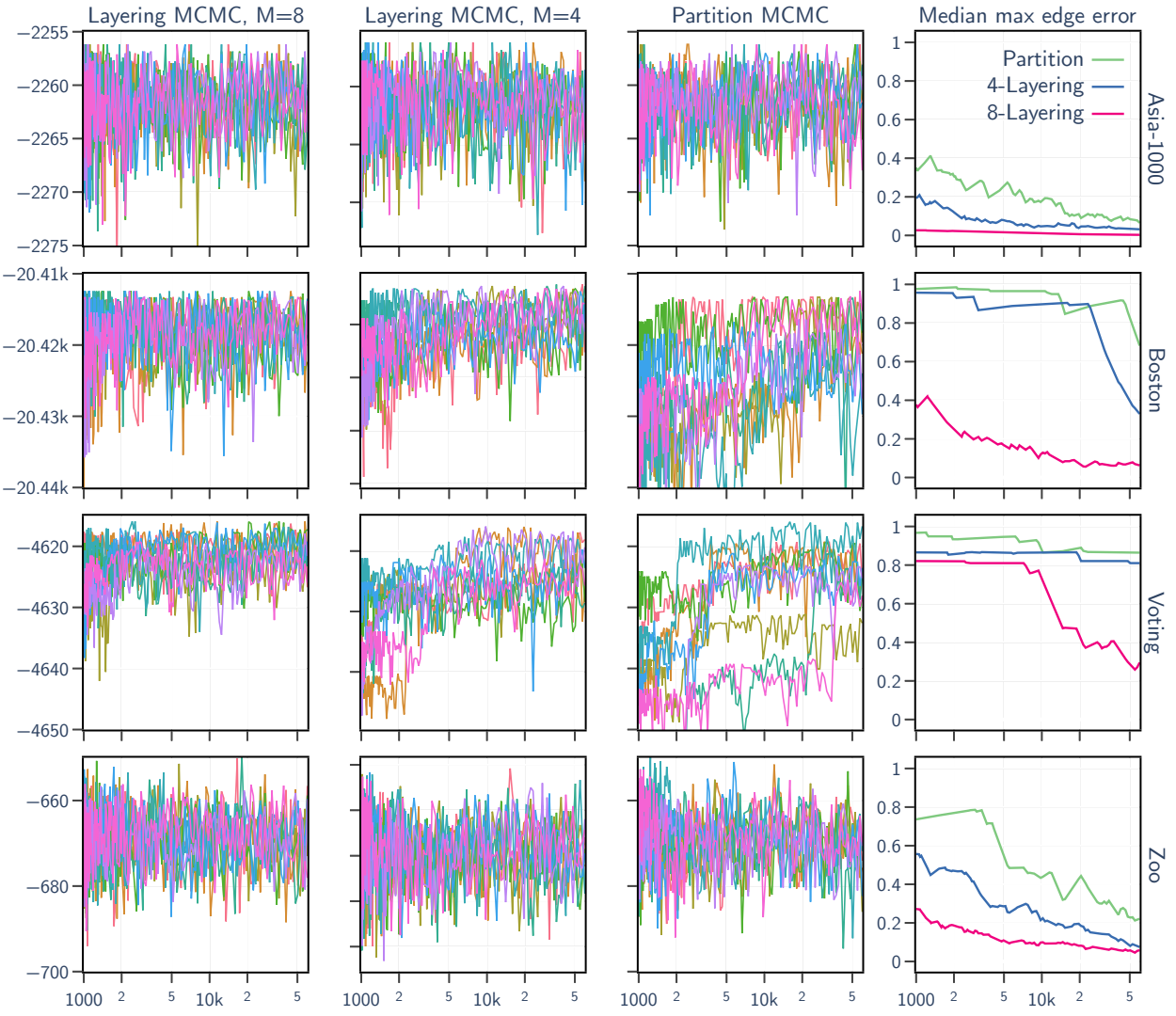

Figure 4: Comparison of layering-MCMC and partition-MCMC on benchmark data sets. *Left:* The posterior probability of the sampled DAG (a logarithm of the unnormalized posterior) per simulation step, in nine independent runs. *Right:* The largest absolute error in the arc posterior probability estimate as a function of the length of the simulation (median over nine independent runs). Note that the x-axis is logarithmic and that, per run, shown are only 200 evenly spaced points out of the 60 000 steps.

that has proven beneficial in partition-MCMC (Kuipers and Moffa, 2017); one only has to ensure that a proposed move is likely to yield a change in the induced layering, to avoid frequently proposing the current state itself. Second, one can equip layering-MCMC with generic boosting techniques, such as Metropolis coupling (Geyer, 1991) and adaptive optimization of proposal distributions (Andrieu and Thoms, 2008). Third, when the number of nodes is in several dozens or in hundreds, one can resort to heuristic pruning of the potential parents sets of a node and, in particular, pre-select a relatively small set of candidate parents (Friedman and Koller, 2003).

Finally, our current implementatation of layering-MCMC is in Python, a high-level programming language, and is unoptimized for speed. We believe that a significant speedup, by two orders of magnitude, can be achieved by using a low-level language like C.

Acknowledgements

This work was partially supported by the Academy of Finland, Grant 316771.

References

Andrieu, C. and Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing*, 18:343–373.

Dua, D. and Graff, C. (2017). UCI machine learning repository. `http://archive.ics.uci.edu/ml`. University of California, Irvine, School of Information and Computer Sciences.

Ellis, B. and Wong, W. H. (2008). Learning causal Bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103:778–789.

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

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

Geyer, C. J. (1991). Markov chain Monte Carlo maximum likelihood. In *Proceedings of the 23rd Symposium on the Interface*, pages 156–163. Interface Foundation of North America.

Grzegorczyk, M. and Husmeier, D. (2008). Improving the structure MCMC sampler for Bayesian networks by introducing a new edge reversal move. *Machine Learning*, 71:265–305.

He, R., Tian, J., and Wu, H. (2016). Structure learning in Bayesian networks of a moderate size by efficient sampling. *Journal of Machine Learning Research*, 17:101:1–101:54.

Kennes, R. (1992). Computational aspects of the Möbius transformation of graphs. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):201–223.

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

Kuipers, J. and Moffa, G. (2017). Partition MCMC for inference on acyclic digraphs. *Journal of the American Statistical Association*, 112:282–299.

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

Lauritzen, S. and Spiegelhalter, D. (1988). Local computation with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 50(2):157–224.

Madigan, D. and York, J. (1995). Bayesian graphical models for discrete data. *International Statistical Review*, 63:215–232.

Malone, B., Kangas, K., Järvisalo, M., Koivisto, M., and Myllymäki, P. (2018). Empirical hardness of finding optimal Bayesian network structures: algorithm selection and runtime prediction. *Machine Learning*, 107(1):247–283.

Niinimäki, T., Parviainen, P., and Koivisto, M. (2016). Structure discovery in Bayesian networks by sampling partial orders. *Journal of Machine Learning Research*, 17:57:1–57:47.

Pensar, J., Talvitie, T., Hyttinen, A., and Koivisto, M. (2020). A Bayesian approach for estimating causal effects from observational data. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*, pages 5395–5402. AAAI Press.

Talvitie, T., Vuoksenmaa, A., and Koivisto, M. (2019). Exact sampling of directed acyclic graphs from modular distributions. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019*, pages 345–354. AUAI Press.

Tian, J. and He, R. (2009). Computing posterior probabilities of structural features in Bayesian networks. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009*, pages 538–547. AUAI Press.