Exact Sampling of Directed Acyclic Graphs from Modular Distributions

Topi Talvitie
University of Helsinki
topi.talvitie@helsinki.fi

Aleksis Vuoksenmaa
University of Helsinki
aleksis.vuoksenmaa@helsinki.fi

Mikko Koivisto
University of Helsinki
mikko.koivisto@helsinki.fi

Abstract

We consider the problem of sampling directed acyclic graphs (DAGs) from a given distribution. We assume the sampling distribution is modular, i.e., the probability of a DAG is a product of local factors, each of which only depends on a node and its parents in the DAG. Using inclusion–exclusion recurrence relations, we give an exact sampler that requires $\tilde{O}(3^n)$ time for preprocessing and $\tilde{O}(2^n)$ per sample, where n is the number of nodes and $\tilde{O}$ suppresses polylogarithmic factors. We also consider the symmetric special case where the factors only depend on the size of the parent set—this covers uniform sampling under indegree constraints. In this case, our exact sampler requires $O(n^3)$ time for preprocessing and $O(n^2)$ per sample; this outperforms the previous best bound even for the uniform distribution. We demonstrate the performance of both samplers also empirically.

1 INTRODUCTION

The protagonist of this paper is the following problem: Given a function w that maps each directed acyclic graph (DAG) on a finite node set V to a non-negative real number $w(G)$, draw a random DAG G with a probability proportional to $w(G)$. We will assume that w is *modular*, i.e., $w(G)$ equals a product of node-wise factors $w_i(G_i)$, $i \in V$, where G_i is the set of parents of node i in G.

This sampling problem arises chiefly in the Bayesian approach to inferring the structure of a Bayesian network (BN) from given data (Koller and Friedman, 2009, Ch. 18.5). The function w is then proportional to a posterior probability function, and obtained by multiplying a modular prior by a modular likelihood function, possibly ignoring a constant factor. The ability to efficiently sample from the posterior would provide a powerful means for accurate, controlled estimation of various quantities of interest. An example of such a quantity is the posterior probability that there is a directed path between two given nodes; for computing this quantity, the fastest known exact algorithm takes time $\tilde{O}(5^n)$, where n is the number of nodes and $\tilde{O}$ hides polylogarithmic factors (Chen et al., 2015).

Prior work has shown that exact sampling from so-called *order-modular*[1] distributions takes time $\tilde{O}(2^n)$, which is nearly linear in the input size (He et al., 2016; Niinimäki et al., 2016). This result stems from stochastic backtracking of the dynamic programming steps for computing the normalizing constant of the distribution (Koivisto and Sood, 2004). For the modular variant, on which we focus in the present paper, the normalizing constant can be computed in time $\tilde{O}(3^n)$ using inclusion–exclusion recurrences (Tian and He, 2009). It is natural to ask whether one can also sample from modular distributions with about the same time complexity.

The state-of-the-art sampling methods for modular distributions are approximate and enjoy only weak accuracy guarantees. Methods based on Markov chain Monte Carlo (Madigan and York, 1995; Kuipers and Moffa, 2017) often perform well in practice, but they offer no reasonable accuracy guarantees for finite runs. On the other hand, there are methods based on importance sampling from the order-modular counterpart of the distribution (Friedman and Koller, 2003; He et al., 2016; Niinimäki et al., 2016); for correcting the sampling bias, the proposed techniques are not completely satisfactory: while one obtains guaranteed lower and upper bounds

---

[1] An order-modular function differs from its modular counterpart by a factor that is proportional to the number of topological sorts of the DAG. Consequently, the class does not include, e.g., uniform distributions, and in the literature, order-modular distributions have been considered mostly for the sake of computational convenience and efficiency.

for the quantity of interest, the interval is wide unless the generated DAGs cover most of the probability mass in the modular distribution, and moreover, computing the interval takes time $\tilde{O}(3^n)$ (He et al., 2016, Thm. 5(iv)).

In this paper, we present an algorithm for exact sampling of DAGs from a given modular distribution in time $\tilde{O}(3^n)$. The algorithm is inspired by the mentioned recurrences of Tian and He (2009), which in turn can be viewed as a generalization of a recurrence for the number of labeled DAGs due to Robinson (1973). From the viewpoint of sampling, the recurrences are of no direct use, however. This is because the involved sum formulas have both positive and negative terms, hence not lending themselves to direct sampling, but rather to (potentially very inefficient) rejection sampling. To overcome this obstacle, we employ the idea of Kuipers and Moffa (2015, 2017) to sample first a so-called *root-layering*, which is a partition of the nodes into layers, and then a DAG conditionally on the root-layering. Our key technical contribution is an efficient algorithm for random generation of root-layerings.

For the fundamental special case where the sampling distribution is uniform (Ide and Cozman, 2002; Melançon et al., 2001; Melançon and Philippe, 2004), fast algorithms are known. Specifically, Kuipers and Moffa (2015) gave an exact sampler that runs in $O(n^5)$ time and a biased sampler that runs in $O(n^2)$ time; here time refers to the number of bit operations. While these samplers are based on efficient random generation of root-layerings, they fall short if one sets constraints, e.g., a maximum number of parents per node; to handle such constraints, Kuipers and Moffa (2015) resort to approximate sampling along a Markov chain. Here, we generalize and improve these samplers: we give an algorithm for exact sampling in time $\tilde{O}(n^3)$ from any symmetric modular distribution, i.e., where each $w_i(G_i)$ only depends on the size $|G_i|$ and not on i; moreover, for the uniform distribution the time requirement is reduced to $\tilde{O}(n^2)$.

The rest of this paper is organized as follows. Section 2 presents our results more formally and also gives an outline of the algorithms. The actual algorithms are presented and analyzed in Sections 3 and 4 for the general and the symmetric case, respectively. Section 5 reports empirical results on the scalability of the samplers.

2   OVERVIEW

This section gives precise formulations of the two problem variants, the model of computation we assume in the time complexity analyses, and our complexity results. We also outline the main ideas underlying the two algorithms, which we describe in detail in the later sections.

2.1   PROBLEMS

It would be natural to formulate the sampling problem in an algebraic model, where the input weights are arbitrary non-negative reals and time complexity amounts to the required number of additions and multiplications. From a practical viewpoint, however, it is crucial that we also pay attention to the cost of representing the input weights and the results of intermediate computations. These considerations motivate us to restrict the input weights to natural numbers, each of which has a finite representation either as a floating-point number (mantissa and exponent part) or as a *plain integer* (no exponent part).

DAG SAMPLING
Input: A set V and maps $w_i : 2^{V \setminus \{i\}} \to \mathbb{N}$ for each $i \in V$.
Output: A random DAG G on V drawn with a probability proportional to $w(G) := \prod_{i \in V} w_i(G_i)$.

Recall that G_i denotes the set of parents of node i in G, i.e., $G_i = \{j : ji \text{ is an edge in } G\}$. Note that the size of the input is exponential in the number of nodes $n := |V|$. Note also that we assume here implicitly that w is not zero everywhere, which could be checked in time $\tilde{O}(3^n)$ by the algorithm of Tian and He (2009).

By restricting the input to maps that are identical and symmetric, i.e., $w_i(S)$ only depends on $|S|$, we arrive at the following problem variant:

SYMMETRIC DAG SAMPLING
Input: A set V and a map $w : \{0, \dots, |V| - 1\} \to \mathbb{N}$.
Output: A random DAG G on V drawn with a probability proportional to $w(G) := \prod_{i \in V} w(|G_i|)$.

Here and henceforth we let w refer to both the "local" input maps and the unnormalized "global" mass function; the particular use will be clear from the context. Note that in this problem variant the input size is linear in $|V|$.

These problems directly generalize to the setting where one is asked to make a given number of independent draws from the distribution. With this setting in mind, we will consider both the *preprocessing* complexity, which is independent of the number of draws, and the *sampling* complexity, which is linear in the number of draws.

2.2   MODEL OF COMPUTATION

Our algorithms operate with large numerical expressions, and the exact representations of their numerical values may grow large. A naive implementation with fixed precision numbers could lead to numerical problems, resulting in incorrect results; the required amount of precision is highly dependent on the algorithm and on the input.

For rigorous treatment of this issue, while maintaining that our algorithms always sample exactly from the correct distribution, we will work under the *word RAM* model (see, e.g., Hagerup, 1998, and references therein), with B_m-bit machine word size. The model assumes that we can perform elementary operations—such as addition, subtraction, multiplication, division, array indexing, and random number generation—on $O(B_\mathrm{m})$-bit integers or floating-point numbers in $O(1)$ time. Furthermore, the model assumes that B_m is large enough to fit indices of input bits (logarithmic in the length of input) and individual input numbers in a single machine word.

2.3 RESULTS

Theorem 1. DAG SAMPLING *can be solved with preprocessing time* $O(3^n n^2)$ *and sampling time* $O(2^n n)$, *assuming the input weights are given as plain integers.*

Here we require that the input weights are plain integers, because our algorithm performs exact integer computations; converting floating-point numbers to integers could blow up the length of the representation. The need of exact arithmetic stems from non-monotonicity: the algorithm employs inclusion–exclusion and is thus susceptible to so-called catastrophic cancellations (i.e., loss of accuracy in subtractions) if only fixed-precision floating-point numbers are used.

For the symmetric variant, the complexity appears to depend on the largest ratio between two input weights,

$$A := \log_2 \Big(\max \big\{ w(a)/w(b) : 1 \le a, b \le n - 1 \big\} \Big);$$

if some input weight is zero, we let $A := \infty$.

Theorem 2. SYMMETRIC DAG SAMPLING *can be solved with preprocessing time* $O\big(n^2 \min\{A + 1, n\}\big)$ *and expected sampling time* $O(n^2)$.

In particular, the preprocessing time is $O(n^2)$ for unconstrained uniform sampling, but the bound becomes $O(n^3)$ for sampling under an indegree constraint. Now, the sampling time is a random variable and we bound its expected value, for the algorithm is based on iteratively increasing the precision of the computations until exact sampling is guaranteed. The input weights can be given as floating-point numbers, because the algorithm uses only numerically stable monotone operations, i.e., additions and multiplications of nonnegative numbers.

For the availability of our implementations of the algorithms, empirical performance results, and discussion of the memory requirements, see Section 5.
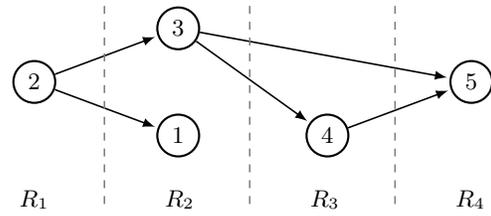

Figure 1: The root-layering $(R_1, \ldots, R_4)$ of a DAG.

2.4 OUTLINE

Both our results (Theorems 1 and 2) stem from the same algorithmic idea: first generating of a random root-layering from the distribution induced by the function $w(G)$, and then a random DAG conditionally on the root-layering. We next introduce the notion of root-layering and give an outline of our algorithms.

For a DAG G, denote by $\rho(G)$ the set of root nodes of G, i.e., $\rho(G) = \{i : G_i = \emptyset\}$. For a set of nodes S in G, denote by $G - S$ the DAG obtained by removing S from G, i.e., the DAG induced by the remaining nodes.

Definition 1. The *root-layering* of a DAG G is the tuple $(R_1, \ldots, R_l)$ where $R_1 = \rho(G)$ and, if $G - R_1$ is not empty, $(R_2, \ldots, R_l)$ is the root-layering of $G - R_1$.

Figure 1 illustrates how the root-layering of a DAG partitions the node set. Observe that the root-layerings of a DAG and its reverse (obtained by reversing the edges of the DAG) generally yield different partitions.

Our algorithms have the following structure:

Step 1 (Preprocess) Compute and store appropriate functions in a dynamic programming fashion.

Step 2 (Sample layering) Draw a random root-layering from the distribution implied by w.

Step 3 (Sample DAG) Draw a random DAG from the conditional distribution given the root-layering.

Given a root-layering, it is relatively easy to draw a random DAG (Step 3) because the choices of the parent sets of different nodes become independent, due to the modularity of the distribution. This is the key observation of Kuipers and Moffa (2015, 2017).

The main challenge is to efficiently draw a root-layering (Step 2). Our key observation is that we can sample iteratively, one layer in turn, starting from the roots of the (random) DAG. While this approach was also taken by Kuipers and Moffa (2015, Sect. 4), there is a crucial difference: since we allow parent set weights that do not

factorize into parent-wise weights, we cannot assign a child's parents independently, but we have to assign the parent set as a whole—this leads to a major difference in the required preprocessing. For the preprocessing we employ inclusion–exclusion recurrences that are similar in spirit to those by Tian and He (2009).

For the symmetric variant, the overall approach is the same; however, monotone computations allow more economical representation of numbers. In particular, we observe that low-precision computations suffice for exact sampling. Furthermore, similarly to Kuipers and Moffa (2015)—but in our more general and exact framework—we exploit the fact that the number of nodes in any layer of a random root-layering is small in expectation.

2.5 NOTATION

We will use the shorthands $R_{a:b} := (R_a, \dots, R_b)$, $\cup R_{a:b} := R_a \cup \dots \cup R_b$, and $\Sigma R_{a:b} := R_a + \dots + R_b$.

We call $R_{1:l}$ a *layering* of U if $R_{1:l}$ is an ordered partition of $U \subseteq V$, and denote by $\mathcal{L}(U)$ the set of layerings of U. We write $G \in R_{1:l}$ as a shorthand for "G is a DAG with root-layering $R_{1:l}$." We denote by $\mathcal{G}_U$ the set of DAGs on U, and by $G[U]$ the subgraph of G induced by U.

3 PROOF OF THEOREM 1

To prove Theorem 1 we will first develop a formula for the conditional probability of one layer in a random root-layering, given the preceding layers. This formula suggests both an efficient way to draw a random root-layering and the auxiliary functions that we need to precompute. After that we will address the complexity of implementing the three steps in detail.

We begin by showing that for a fixed root-layering, the choices of the parent sets are independent of each other. In fact, we will need a stronger characterization that shows how the choices for one layer only depend on the previous layer and the *union* of the preceding layers. To this end, for any pair of sets $R \subseteq T$, define the *collection of candidate parents sets* as

$$\mathcal{C}(R, T) := \{S \subseteq T : S \cap R \neq \emptyset \text{ or } R = \emptyset\}. \quad (1)$$

Lemma 3. *Let $R_{1:l} \in \mathcal{L}(V)$ and $R_0 = \emptyset$. A digraph G on V is a DAG with root-layering $R_{1:l}$ if and only if $G_i \in \mathcal{C}(R_{k-1}, \cup R_{1:k-1})$ for all $1 \leq k \leq l$ and $i \in R_k$.*

Proof. We prove by induction on the set V. The claim trivially holds for $V = \emptyset$. For the induction step, assume the claim holds for all proper subsets of V.

Let G be a DAG on V with root-layering $R_{1:l}$. If $i \in R_1$, then i is a root of G; hence, $G_i = \emptyset \in \mathcal{C}(\emptyset, \emptyset)$. For

the remaining layers, apply the induction assumption for the subgraph G' of G induced by $V \setminus R_1$. Because $R_{2:l}$ is the root-layering of G', we have for all $i \in R_2$ that $G'_i \in \mathcal{C}(\emptyset, \emptyset)$. Thus, G'_i is empty and $G_i \subseteq R_1$. As i is a non-root of G, the set G_i intersects R_1; hence, $G_i \in \mathcal{C}(R_1, R_1)$, as desired. If $i \in R_k$ with $k \geq 3$, then $G'_i \in \mathcal{C}(R_{k-1}, \cup R_{2:k-1})$, whence $G_i \in \mathcal{C}(R_{k-1}, \cup R_{1:k-1})$.

For the other direction, assume $G_i \in \mathcal{C}(R_{k-1}, \cup R_{1:k-1})$ for all $1 \leq k \leq l$ and $i \in R_k$. It follows that G_i is empty exactly when $i \in R_1$, and thus R_1 must consist of the roots of G. Now, for each $i \in V \setminus R_1$, put $G'_i := G_i \setminus R_1$. If $i \in R_2$, then $G_i \subseteq R_1$ and thus $G'_i = \emptyset \in \mathcal{C}(\emptyset, \emptyset)$. If $i \in R_k$, $k \geq 3$, then G_i intersects R_{k-1} and is contained in $\cup R_{2:k-1}$; hence, $G'_i \in \mathcal{C}(R_{k-1}, \cup R_{2:k-1})$. By the induction assumption, G' is a DAG with root-layering $R_{2:l}$. As G is obtained from G' by adding the nodes in R_1 and edges from R_1 to $\cup R_{2:l}$ such that all roots of G are in R_1, also G is a DAG with root-layering $R_{1:l}$. $\square$

By Lemma 3, the probability that a random DAG has root-layering $R_{1:l}$ is proportional to

$$\sum_{G \in R_{1:l}} \prod_{i \in V} w_i(G_i) = \prod_{k=1}^{l} \prod_{i \in R_k} \hat{w}_i(R_{k-1}, \cup R_{1:k-1}), \quad (2)$$

where $\hat{w}_i(R, T) = \sum_{S \in \mathcal{C}(R,T)} w_i(S)$. Indeed, the sum can be written as n nested sums, in each of which G_i runs over $\mathcal{C}(R_{k-1}, \cup R_{1:k-1})$, with $i \in R_k$ and $R_0 = \emptyset$. Reorganizing into a product of sums yields (2).

The conditional probability that the p:th layer is R_p, given that the preceding layers are $R_{1:p-1}$ and the remaining node set $U := V \setminus \cup R_{1:p-1}$ is not empty, is proportional to the joint probability of $R_{1:p}$. This probability, in turn, is the marginal of the joint probability of $R_{1:l}$, obtained by taking a sum of the product (2) over the possible suffixes $R_{p+1:l}$. Now, eliminate the factors that do not depend on R_p and take the common factor, for $k = p$, out of the sum. We get that the desired conditional probability is proportional to

$$f(R_p, U) \prod_{i \in R_p} \hat{w}_i(R_{p-1}, V \setminus U), \quad (3)$$

where for all $\emptyset \neq I_0 \subseteq U \subseteq V$ we define

$$f(I_0, U) := \sum_{I_{1:s} \in \mathcal{L}(U \setminus I_0)} \prod_{k=1}^{s} \prod_{i \in I_k} \hat{w}_i(I_{k-1}, V \setminus \cup I_{k:s}). \quad (4)$$

Using formula (3), we can generate a random root-layering, provided that the f- and $\hat{w}_i$-values are available. Next we show how they can be efficiently precomputed, and after that we consider the DAG sampling step.

3.1 PRECOMPUTATION

Consider first the computation of the function $\hat{w}_i$ for a fixed $i \in V$. Observe that for all $T \subseteq V \setminus \{i\}$, we can compute $\hat{w}_i(\emptyset, T)$ and $\hat{w}_i(\{x\}, T)$ for all $x \in T$ by simple summations over the w_i-values, using $O(3^n)$ additions in total. For the rest of the $\hat{w}_i(R, T)$-values, we can express $\mathcal{C}$ recursively as a disjoint union:

$$\mathcal{C}(R, T) = \mathcal{C}(\{x\}, T) \cup \mathcal{C}(R \setminus \{x\}, T \setminus \{x\}), \quad (5)$$

where $R \subseteq T$ such that $|R| \geq 2$ and x is an arbitrary element of R. For the $\hat{w}_i$-values, this translates to the recurrence $\hat{w}_i(R, T) = \hat{w}_i(\{x\}, T) + \hat{w}_i(R \setminus \{x\}, T \setminus \{x\})$. Using this recurrence, precomputing all the $\hat{w}_i$-values for all $i \in V$ requires $O(3^n n)$ additions.

Remark 1. Alternatively, we can first compute $\hat{w}_i(\emptyset, T)$ using fast zeta transforms, with $O(2^n n^2)$ addition operations in total, and then the rest using the identity $\hat{w}_i(R, T) = \hat{w}_i(\emptyset, T) - \hat{w}_i(\emptyset, T \setminus R)$. But this would not affect the total precomputation time complexity.

Next, consider computing the f-values. Define for all $R \subseteq U \subseteq V$ the DAG collections

$$\mathcal{G}(R, U) := \left\{ G \in \mathcal{G}_V : \rho(G) \supseteq V \setminus U, \rho(G[U]) = R \right\},$$
$$\bar{\mathcal{G}}(R, U) := \left\{ G \in \mathcal{G}_V : \rho(G) \supseteq V \setminus U, \rho(G[U]) \supseteq R \right\}.$$

In words, both collections only contain DAGs G on V where every node in $V \setminus U$ is a root. The difference is that in the induced subgraph $G[U]$ exactly the nodes in R are the roots for a $G \in \mathcal{G}(R, U)$, whereas for a $G \in \bar{\mathcal{G}}(R, U)$ also some node in $U \setminus R$ can be a root.

The DAGs on V where all $V \setminus U$ are roots are exactly the DAGs that can be obtained from a DAG on U by adding edges from $V \setminus U$ to U. Thus, by Lemma 3 and definition (4) we get a new interpretation for $f(R, U)$:

$$f(R, U) = \sum_{G \in \mathcal{G}(R, U)} \prod_{i \in U \setminus R} w_i(G_i). \quad (6)$$

By the principle of inclusion and exclusion,

$$f(R, U) = \sum_{R \subseteq X \subseteq U} (-1)^{|X \setminus R|} \sum_{G \in \bar{\mathcal{G}}(X, U)} \prod_{i \in U \setminus R} w_i(G_i).$$

The set $\bar{\mathcal{G}}(X, U)$ consists of the DAGs that can be obtained from a DAG in $\bar{\mathcal{G}}(\emptyset, U \setminus X)$ by changing the initially empty parent set of each node in X to a subset of $V \setminus U$. Thus we get that

$$f(R, U) = \sum_{R \subseteq X \subseteq U} (-1)^{|X \setminus R|} g(U \setminus X) \prod_{i \in X \setminus R} \hat{w}_i(\emptyset, V \setminus U),$$

where

$$g(U) := \sum_{G \in \bar{\mathcal{G}}(\emptyset, U)} \prod_{i \in U} w_i(G_i).$$

This formula for f is still problematic from the complexity viewpoint because computing it directly would require considering $\Omega(4^n)$ triplets (R, U, X). To reduce the complexity, consider a fixed $U \subseteq V$ and the reparameterized slice $f_U(S) := f(U \setminus S, U)$ for all $S \subseteq U$. Substituting $X := U \setminus Y$ yields

$$f_U(S) = \sum_{Y \subseteq S} (-1)^{|S \setminus Y|} g(Y) \prod_{i \in S \setminus Y} \hat{w}_i(\emptyset, V \setminus U).$$

Now, observe that f_U, when viewed as a vector indexed by S, is a product of a $2^{|U|} \times 2^{|U|}$ matrix K and the vector g of length $2^{|U|}$; the matrix entries are given by

$$K_{SY} := \prod_{i \in U} \left(- \hat{w}_i(\emptyset, V \setminus U) \right)^{[i \in S \setminus Y \text{ and } Y \subseteq S]},$$

where $[Q]$ evaluates to 1 if Q holds, else to 0. Thus, K is a Kronecker product of $|U|$ matrices of size 2×2, implying that f_U can be computed with $O\left(2^{|U|} |U|\right)$ additions and multiplications (Yates, 1937; Kennes, 1992).

It remains to compute the g-values. Using the inclusion–exclusion principle, we obtain for all $\emptyset \neq U \subseteq V$ that

$$g(U) = \sum_{\emptyset \neq R \subseteq U} (-1)^{|R|+1} \sum_{G \in \bar{\mathcal{G}}(R, U)} \prod_{i \in U} w_i(G_i).$$

Again, by relating the DAGs in $\bar{\mathcal{G}}(R, U)$ to those in $\bar{\mathcal{G}}(\emptyset, U \setminus R)$, we get a recursive formula:

$$g(U) = \sum_{\emptyset \neq R \subseteq U} (-1)^{|R|+1} g(U \setminus R) \prod_{i \in R} \hat{w}_i(\emptyset, V \setminus U)$$

with the base case $g(\emptyset) = 1$.

Proposition 4. *Computing the values $f(R, U)$ for all $\emptyset \neq R \subseteq U \subseteq V$ and $\hat{w}_i(R, T)$ for all $i \in V$ and $R \subseteq T \subseteq V \setminus \{i\}$ requires $O(3^n n^2)$ time.*

Proof. As the input size is $\Omega(2^n)$, the machine word size B_{m} is $\Omega(n)$. Furthermore, each subset of V can be represented by $O(1)$ machine words, and set operations on them take $O(1)$ time.

When computing the $\hat{w}_i$-values, each addition is with numbers smaller than $2^n \cdot 2^{B_{\mathrm{m}}}$. Thus the operands fit in $O(1)$ machine words and the running time is $O(3^n n^2)$.

In the formulas for f and g, each number fits in $O(n)$ machine words, since the numbers do not exceed $|\bar{\mathcal{G}}(\emptyset, V)| \cdot 2^{B_{\mathrm{m}} n} \leq 2^{n^2 + B_{\mathrm{m}} n} = O(2^{B_{\mathrm{m}} n})$. By ordering the summation over $\emptyset \neq R \subseteq U$ in the recurrence for g by increasing R and reusing the subproducts from earlier terms, we use $O(n)$ time for each term, as the second operand in each multiplication operation is a $\hat{w}$-value and thus fits in $O(1)$ words. Because there are $O(3^n)$ terms in total, computing all the g-values takes $O(3^n n)$ time.

Using fast algorithms for generalized Möbius transform (Yates, 1937; Kennes and Smets, 1990; Kennes, 1992), computing f_U requires $O\big(2^{|U|}|U|\big)$ additions and multiplications of numbers that fit in $O(n)$ words, and again in each multiplication the second operand fits in $O(1)$ words. As we need f_U for all $U \subseteq V$, computing all the f-values takes $O(3^n n^2)$ time. □

3.1.1 Monotone Version

The inclusion–exclusion algorithm requires subtraction, which can lead to numerical problems if implemented using fixed precision. We next give alternative precomputation formulas that only add and multiply nonnegative numbers. We will build on these formulas in Section 4.

The algorithm for computing the $\hat{w}_i$-values uses only addition operations, so it requires no changes. To compute the f-values without subtraction, consider the definition (4), and separate the summation over I_1 to the front. We get that $f(I_0, U)$, for $I_0 \subsetneq U \subseteq V$, can be written as

$$\sum_{\emptyset \neq I_1 \subseteq U \setminus I_0} \left[\prod_{i \in I_1} \hat{w}_i\big(I_0, V \setminus (U \setminus I_0)\big) \right.$$

$$\left. \times \sum_{I_{2:s} \in \mathcal{L}(U \setminus (I_0 \cup I_1))} \prod_{k=2}^{s} \prod_{i \in I_k} \hat{w}_i(I_{k-1}, V \setminus \cup I_{k:s}) \right].$$

Reindexing the second factor inside the summation to match the formula (4) for $f(I_1, U \setminus I_0)$ gives us that

$$f(I_0, U) = \sum_{\emptyset \neq I_1 \subseteq U \setminus I_0} f(I_1, U \setminus I_0) \prod_{i \in I_1} \hat{w}_i\big(I_0, V \setminus (U \setminus I_0)\big).$$

The base case is $f(U, U) = 1$ (by applying (4) directly). To compute $f(I_0, U)$ for all $\emptyset \neq I_0 \subseteq U \subseteq V$ via the recurrence, we need to consider $O(4^n)$ sum terms in total, since there are $O(4^n)$ triplets (I_0, I_1, U) satisfying $I_0 \cap I_1 = \emptyset$ and $I_0 \cup I_1 \subseteq U \subseteq V$. Summing over $\emptyset \neq I_1 \subseteq U \setminus I_0$ in increasing order by $|I_1|$ and reusing the subproducts from the previous terms, each term requires $O(1)$ multiplications and additions.

Remark 2. The monotone algorithm can efficiently operate with floating-point numbers. Using $O(1)$-word numbers it requires $O(4^n)$ time for precomputation and achieves very small maximum relative error. The error can be mitigated by adaptively rerunning the precomputation if the precision is insufficient for exact sampling. We do this for the symmetric version in Section 4.3; the proofs can be adapted to the asymmetric case.

3.2 SAMPLING

After precomputation, for each sample we draw we need to do two things: sample a root-layering $R_{1:l}$ and then sample a DAG conditional on the root-layering.

Recall that we sample the root-layering iteratively: having $R_{1:p-1}$ sampled and $U := V \setminus \cup R_{1:p-1}$ nonempty, a set $\emptyset \neq R_p \subseteq U$ is chosen as the next layer with probability proportional to $f(R_p, U) \prod_{i \in R_p} \hat{w}_i(R_{p-1}, V \setminus U)$, where $R_0 := \emptyset$. When sampling R_p, we have to consider $2^{|U|} - 1 \leq 2^{n+1-p}$ different sets, which means we consider at most $2^n + 2^{n-1} + 2^{n-2} + \ldots = O(2^n)$ sets in total. By considering the different possibilities for R_p ordered by inclusion and reusing the subproducts from earlier sets, we need only $O(1)$ multiplications for computing each weight. Considering the set R_p in increasing order by size and reusing the subproducts from earlier sets, $O(1)$ multiplications suffice per R_p.

Once the root-layering $R_{1:l}$ is fixed, by Lemma 3, we can sample the DAG simply by drawing for all $1 \leq k \leq l$ and $i \in R_k$ the parent set G_i from $\mathcal{C}(R_{k-1}, \cup R_{1:k-1})$ weighted by w_i. If we order all the nodes by the root-layering, then for each node we consider only subsets of its predecessors, and thus we consider at most $2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots = O(2^n)$ subsets in total.

All arithmetic operations are with numbers that fit in $O(n)$ machine words, and in each multiplication the second operand fits in $O(1)$ words (cf. proof of Prop. 4). Thus we get that after precomputation, each sample can be drawn in $O(2^n n)$ time. This, together with Proposition 4, completes the proof of Theorem 1.

Remark 3. We could sample in polynomial time by investing $\tilde{O}(4^n)$ time to precomputation: by computing the probability distribution over the next layer R_p for each pair (R_{p-1}, U) and the distribution for $G_i \in \mathcal{C}(R_{k-1}, P)$ weighted by w_i for each triplet (i, R_{k-1}, P).

4 PROOF OF THEOREM 2

To prove Theorem 2, we will adapt the algorithms of Section 3 by replacing node sets by their sizes: $\hat{w}_i(R, T)$, $f(R, U)$ become $\hat{w}(r, t)$, $f(r, u)$, respectively. We will build on the monotone version of the algorithm, introducing several modifications.

4.1 NUMERICAL PRECISION

As our algorithms will no longer be exponential time, we have to consider the complexity of the numeric operations with greater detail. Because we now allow nonnegative floating-point weights, our computations are inexact. To enable exact sampling, we will use variable precision: first do all computations with floating-point numbers with mantissa length of $O(1)$ machine words, keeping track of the maximum error; as long as the result is inconclusive, double the mantissa length and redo the computations using the same random number sequence.

To bound the numerical errors in our computations, we define[2] the relative error of an approximation x' of x as $|\ln x' - \ln x|$. The maximum relative error of floating-point operations with L-word mantissa is the *machine epsilon* $\epsilon_{\mathrm{m}}(L)$, or ϵ_{m} for short, equal to $\ln(1 + 2^{-LB_{\mathrm{m}}})$. If a' and b' are floating-point representations of a and b with relative errors ϵ_a and ϵ_b, then the relative error of the floating-point addition or multiplication of a' and b' is at most $\max\{\epsilon_a, \epsilon_b\} + \epsilon_{\mathrm{m}}$ or $\epsilon_a + \epsilon_b + \epsilon_{\mathrm{m}}$, respectively.

The exponent parts in the floating-points numbers will always fit in $O(1)$ machine words and thus not affect the time complexity. (Indeed, all the computed numbers are sums of products of at most n input weights, with at most $\exp(O(n^2))$ terms. Thus the logarithms of the (absolute) values get increased by a factor at most $O(n^2)$, implying that the lengths of the exponent part get increased by at most $O(\log n)$ bits, staying within $O(1)$ machine words.)

4.2 PRECOMPUTATION

In order to compute the $\hat{w}$-values, we first compute for all $1 \leq t \leq n-1$ the entries

$$\hat{w}(0,t) = \sum_{j=0}^{t} \binom{t}{j} w(j), \ \hat{w}(1,t) = \sum_{j=1}^{t} \binom{t-1}{j-1} w(j);$$

we precompute the binomial coefficients via the Pascal triangle. The rest, for $2 \leq r \leq t \leq n-1$, we compute via the recurrence $\hat{w}(r,t) = \hat{w}(1,t) + \hat{w}(r-1,t-1)$.

To compute $f(r,u)$ for all $0 \leq r \leq u \leq n$, we employ the symmetric version of the monotone recurrence:

$$f(r,u) = \sum_{x=1}^{u-r} \binom{u-r}{x} f(x, u-r)\, \hat{w}(r, n-u+r)^x \ . \quad (7)$$

To state the time complexity of the precomputation, let M_L denote the number of operations needed for multiplying two L-word numbers. Known results for integer multiplication imply that $M_L = \tilde{O}(L)$.

Proposition 5. *Using L-word floating-point precision, we can compute the values $f(r,u)$ for all $1 \leq r \leq u \leq n$ and $\hat{w}(r,t)$ for all $0 \leq r \leq t \leq n-1$ in $O(n^3 M_L)$ time to within a relative error of $O(n^4 \epsilon_{\mathrm{m}})$.*

Proof. The binomial coefficients and $\hat{w}$-values can be computed with $O(n^2)$ additions. As each addition increases the maximum relative error by at most ϵ_{m}, the maximum relative error of these values is $O(n^2 \epsilon_{\mathrm{m}})$.

We compute the powers $\hat{w}(r, n-u+r)^x$ for all relevant r, u, x by repeated multiplication, requiring $O(n^3)$

multiplications and resulting in $O(n^3 \epsilon_{\mathrm{m}})$ relative error. After that, we compute $f(r,u)$ for all $1 \leq r \leq u \leq n$ in increasing order by u with $O(n^3)$ multiplications and additions, using the recurrence for f. We can prove by induction on u that the relative error of $f(r,u)$ is at most $C\, u\, n^3\, \epsilon_{\mathrm{m}}$ for some constant C; thus the maximum relative error of the output values is $O(n^4 \epsilon_{\mathrm{m}})$.

In total, we do $O(n^3)$ arithmetic operations with L-word floating-point numbers, thus taking $O(n^3 M_L)$ time. $\square$

4.3 SAMPLING

Similarly to the general case, we sample the layer sizes of the root-layering, $r_{1:l}$, iteratively: having $r_{1:p-1}$ sampled and $u := n - \Sigma r_{1:p-1}$ nonzero, a number $1 \leq r_p \leq u$ is chosen as the size of the next layer with probability proportional to $\binom{u}{r_p} f(r_p, u) \hat{w}(r_{p-1}, n-u)^{r_p}$, where $r_0 = 0$. Due to symmetry, we can then sample the root-layering $R_{1:l}$ by distributing the elements of V into the sets such that $|R_k| = r_k$ for all $1 \leq k \leq l$ uniformly at random.

Once the root-layering $R_{1:l}$ is fixed, by Lemma 3, we can sample the DAG by drawing for all $1 \leq k \leq l$ and $i \in R_k$ the parent set G_i from $\mathcal{C}(R_{k-1}, \cup R_{1:k-1})$ weighted by $w(|G_i|)$. If $k = 1$, then always $G_i = \emptyset$. Suppose $k \geq 2$ and define $P := \cup R_{1:k-1}$. Now, apply (5) repeatedly to the nodes in $x \in R_{k-1}$ along some total order $\prec$ on V to partition $\mathcal{C}(R_{k-1}, P)$ into a disjoint union of the sets $\mathcal{C}(\{x\}, P_x)$, where $P_x = P \setminus \{y \in R_{k-1} : y \prec x\}$. The idea is to first choose the smallest $x \in R_{k-1}$ contained in G_i, and then choose the rest of G_i from P_x. The probability of choosing x is proportional to $\hat{w}(1, |P_x|)$. To draw the rest of G_i, draw first the size j of G_i with probability proportional to $\binom{|P_x|-1}{j-1} w(j)$, with $1 \leq j \leq |P_x|$, and then j nodes from $P_x \setminus \{x\}$ uniformly at random.

Because the computed numbers are (controlled) approximations, we have to check whether they suffice for exact sampling. We can detect if they do not, and then call the attempt a failure and repeat the computations with increased precision parameter L, using the same random numbers. The following result concerns a single trial:

Proposition 6. *After precomputation with L-word precision, the DAG can be sampled in $O(n^2 M_L)$ time with failure probability at most $\exp\left(O(n^7 \epsilon_{\mathrm{m}})\right) - 1$.*

Proof. The sampling algorithm consists of several steps, in each of which we draw some j from $\{1, 2, \ldots, m\}$ with probability proportional to p_j. We implement the sampling step by computing the cumulative sums $\Sigma p_{1:j}$ for all $0 \leq j \leq m$, drawing a number $t \in [0, \Sigma p_{1:m})$ uniformly at random and finding the first $1 \leq j \leq m$ such that $t < \Sigma p_{1:j}$. Thus the sampling phase requires $O(m)$ addition and comparison operations.

---

[2]Our definition deviates from the standard $|x' - x|/x$. Ours is convenient for estimating the error of multiplications. The two definitions are approximately equal when the error is small.

Both sampling the layer sizes and the parent set sizes require $O(n)$ draws. For each draw there are $O(n)$ outcomes, whose unnormalized probabilities are computed with $O(n)$ additions and multiplications (using repeated multiplication for the powers of $\hat{w}(r_{p-1}, n-u)$ when sampling the root-layering). Sampling the parent sets when their sizes are fixed takes $O(n^2)$ time, for that can be implemented using $O(\log n)$-bit exact integer arithmetic. In total, the algorithm runs in $O(n^2 M_L)$ time.

By the error bounds for the precomputed values (Prop. 5), we get the bound $O(n^5 \epsilon_m)$ for the relative errors of the sampling proportions. If we normalize the proportions to probabilities, all values are in range $[0,1]$, and thus their absolute errors are $\exp(O(n^5 \epsilon_m)) - 1$. Thus in each sampling step, the probability that there is not enough precision to determine the correct result is $\exp(O(n^6 \epsilon_m)) - 1$. As there are $O(n)$ sampling steps, the total failure probability is $\exp(O(n^7 \epsilon_m)) - 1$. $\qquad\square$

Now, consider the expected running time of the full exact sampling method in which we increase the precision parameter L until sampling succeeds. We run the precomputation for a fixed initial precision L', which is a constant independent of n. Then, in the sampling phase, we first try sampling with $L = L'$, and as long as it fails, we keep doubling the precision, each time running both the precomputation and sampling again with precision $L = 2^F L'$, where F is the number of failures so far.

By Propositions 5 and 6, we know that (for some choice of time unit) the time complexities of precomputation and sampling are at most $n^3 M_L$ and $n^2 M_L$, respectively, and the failure probability is at most $\exp\left[cn^7\epsilon_m(L)\right] - 1$ for some constant $c > 0$. Thus, we get that the expected sampling time is bounded from above by

$$n^2 M_{L'} + \sum_{F=1}^{\infty} n^3 M_{2^F L'} \left(\exp\left[cn^7\epsilon_m(2^{F-1}L')\right] - 1 \right).$$

It suffices to show that the sum over F, which we denote by S, satisfies $S \leq n^2$ for some choice of L' that is independent of n. Indeed, this will imply that with $O(n^3)$-time precomputation we can sample DAGs in $O(n^2)$ expected time, proving Theorem 2 for the case $A = \Omega(n)$.

We may assume w.l.o.g. that $n \geq c + 4$. Put $L' := 16$ and apply the inequalities $M_{2^F L'} \leq 4^{F+4} \leq n^{F+4}$ and $\epsilon_m(L) \leq 2^{-LB_m} \leq n^{-L}$ to get that

$$S \leq \sum_{F=1}^{\infty} n^{7+F} \left(\exp\left[n^{8-2^{F+3}}\right] - 1 \right).$$

Since $\exp(x) - 1 \leq 2x$ for $0 \leq x \leq 1$, we have that

$$S \leq n^2 \sum_{F=1}^{\infty} 2\, n^{13+F-2^{F+3}}.$$

Finally, observe that $13 + F - 2^{F+3} \leq -F$ for all $F \geq 1$.

Remark 4. In practical settings, the exact approach is often unnecessary: using large enough fixed precision, the failure probability can be made insignificant. The failure probability bound in Proposition 5 is loose; tracking the maximum numerical errors in run time is a practical way to verify the exactness of sampling.

4.4 NUMBER OF ROOTS

To finish the proof of Theorem 2, we will make use the observation that an overwhelming majority of DAGs have very few root nodes. A similar idea was followed by Kuipers and Moffa (2015) for sampling DAGs from a slightly biased uniform distribution with only $O(n^2)$ bit operations. In our case, the distribution is not uniform and the weights will affect the performance of the method through the parameter A, which measures the uniformity of the weights. Moreover, we want to sample exactly from the correct distribution.

For a probabilistic bound on the number of root nodes, we build on the following result due to Liskovets (1975): for $2 \leq \ell \leq m$, the probability that a DAG with m labeled nodes chosen uniformly at random has more than ℓ roots is at most

$$\frac{2(m-\ell)}{(m+1)(\ell+1)!\, 2^{\ell(\ell-1)/2}} \leq 2^{-\ell(\ell-1)/2}. \qquad (8)$$

We will next extend this results to our case of weighted DAGs. Moreover, we want to bound the sizes of all layers in the root-layering, not just the first.

Recall that for all $\emptyset \neq R \subseteq U \subseteq V$ we defined the set

$$\mathcal{G}(R,U) = \{G \in \mathcal{G}_V : \rho(G) \supseteq V \setminus U, \rho(G[U]) = R\}.$$

Observe that we can write the probability that a random DAG with $|U|$ labeled nodes has more than ℓ roots as the ratio $\sum_{|X|>\ell} |\mathcal{G}(X,U)| / \sum_X |\mathcal{G}(X,U)|$, where the sums are over $\emptyset \neq X \subseteq U$; this is because the choice of edges from $V \setminus U$ to U is independent of the structure of the induced subgraph $G[U]$. We can also write this ratio using the formula (6) we developed for the asymmetric weighted case, *provided that we set all weights to* 1, as

$$\frac{\sum_{\emptyset \neq X \subseteq U, |X|>\ell} f(X,U) \prod_{i \in X} \hat{w}_i(\emptyset, V \setminus U)}{\sum_{\emptyset \neq X \subseteq U} f(X,U) \prod_{i \in X} \hat{w}_i(\emptyset, V \setminus U)}.$$

Now, translating to the symmetric case, under the same assumption ($w(s) = 1$ for all $0 \leq s \leq n-1$), and using Liskovets's inequality (8), yields

$$\frac{\sum_{x=\ell+1}^{u} \binom{u}{x} f(x,u)\hat{w}(0,n-u)^x}{\sum_{x=1}^{u} \binom{u}{x} f(x,u)\hat{w}(0,n-u)^x} \leq 2^{-\ell(\ell-1)/2}.$$

We want to apply this inequality to prove that for some ℓ, we introduce only negligible error by using only the first ℓ terms in the sum of the recurrence (7) for computing the f-values. Our problem is, however, that the terms in the recurrence are of type $\binom{u}{x} f(x,u)\hat{w}(r, n-u)^x$, where $1 \le r \le n-u$, whereas the inequality we obtained is only valid for $r = 0$. We can fix this by observing that because the weight is uniform, it holds that $\hat{w}(0, n-u)/2 \le \hat{w}(r, n-u) \le \hat{w}(0, n-u)$, and thus

$$p_\ell := \frac{\sum_{x=\ell+1}^{u} \binom{u}{x} f(x,u)\hat{w}(r, n-u)^x}{\sum_{x=1}^{u} \binom{u}{x} f(x,u)\hat{w}(r, n-u)^x} \le 2^{n-\ell(\ell-1)/2}.$$

Now, let us remove the assumption of uniform weights and denote by A the binary logarithm of the ratio between the maximum and minimum weight (in the case of at least one zero weight, $A = \infty$). Because both the dividend and the divisor in p_ℓ are homogeneous polynomials of degree u in the input weights, removing the uniformity assumption increases p_ℓ by a factor at most $2^{uA} \le 2^{nA}$. Thus $p_\ell \le 2^{n(A+1)-l(\ell-1)/2}$ for all $2 \le \ell \le u$.

To ensure that the error is negligible, i.e., $p_\ell = O(\epsilon_{\mathrm{m}})$, it is sufficient to set $\ell = O\big(\sqrt{n(A+1) + B_{\mathrm{m}}}\big)$, because $\epsilon_{\mathrm{m}} = \Theta(2^{-LB_{\mathrm{m}}})$ and the precision parameter L is a constant in the precomputation phase. The introduced $O(\epsilon_{\mathrm{m}})$ relative error for the f-values is dominated by the error bound $O(n^4 \epsilon_{\mathrm{m}})$ of Proposition 5. Furthermore, we only need to compute a value $f(r, u)$ if $r \le \ell$, because in the sampling phase we can use a similar argument to show that we can ignore the layer sizes r_p larger than ℓ. This improves the precomputation time to $O(n^2 + n\ell^2)$. In the rare case that this optimization changes the result in the sampling phase, the sampling will fail. If that happens, we retry the sampling with higher precision similarly to Section 4.3, without the optimization, and thus we always get the correct result.

For the complexity analysis of the algorithm, we may assume that $B_{\mathrm{m}} = \Theta(\log n)$, because in the previous results we only needed that $B_{\mathrm{m}} = \Omega(\log n)$ and rounding input values to the used precision introduces at most ϵ_{m} relative error, which does not affect the analysis. Thus setting $\ell = O\big(\sqrt{n(A+1) + \log n}\big) = O\big(\sqrt{n(A+1)}\big)$ is sufficient, which means that the precomputation time complexity is $O(n^2(A+1))$. This combined with the result of Section 4.3 completes the proof of Theorem 2.

5 PRACTICE

We have implemented several variants of the presented algorithms into a publicly available C++ program.[3] Performance tests were run on a desktop computer with an Intel Core i7-4790 processor and 16 GB of memory.

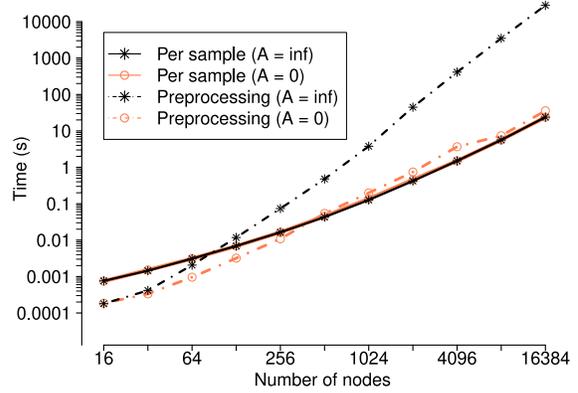[3] github.com/ttalvitie/modular-dag-sampling

Figure 2: Running times for sampling from symmetric distributions. Shown are averages over 10 runs.

In the asymmetric case, it is feasible to sample DAGs on up to around $n = 20$ nodes due to the exponential time and memory requirements. In this range, the monotone algorithm is, in fact, faster than the asymptotically faster inclusion–exclusion algorithm, both in theory and in practice ($4^n < 3^n n^2$ if $n < 22$). Following Remarks 2 and 4, we tested a fixed-precision implementation of the monotone algorithm. With $n = 15$ nodes and 7-bit random input weights, preprocessing took 119 seconds and drawing one DAG took 0.031 seconds; this required 90 % of the memory. Extrapolating from this, it would take around 2 days to sample a 20-node DAG, provided that a few terabytes of memory is available.

We also tested a variant that trades time for space. The space requirement is dominated by the storage of the $\hat{w}_i$-values. To reduce the requirement, we only precomputed $\hat{w}_i(R, U)$ for $|R| \le 1$ and employed the recurrence given in Section 3.1 to compute $\hat{w}_i(R, U)$ for $|R| > 1$ when needed. This increases the time requirement by a $O(n)$ factor. With this modification, we were able to handle 17-node DAGs, spending about 5 hours (18 600 seconds) for preprocessing and 0.270 seconds per sample.

In the symmetric case, we tested the presented algorithm with and without the optimization based on small expected layer sizes. We observed (Figure 2) that both the required preprocessing time and the required sampling time are in line with the asymptotic (cubic and quadratic) bounds given in Theorem 2. For comparison, Kuipers and Moffa (2015) report their exact uniform sampler (implemented in Maple) being able to draw six 100-node DAGs in a second (preprocessing time excluded). Our sampler appears to be two orders of magnitude faster—we suspect this to be partly because of the different programming languages, but also because of the difference in the asymptotic time complexities of the samplers.

References

Chen, Y., Meng, L., and Tian, J. (2015). Exact Bayesian learning of ancestor relations in Bayesian networks. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 174–182. PMLR.

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.

Hagerup, T. (1998). Sorting and searching on the word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 366–398. Springer.

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.

Ide, J. S. and Cozman, F. G. (2002). Random generation of Bayesian networks. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence*, volume 2507 of *Lecture Notes in Computer Science*, pages 366–375. Springer.

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

Kennes, R. and Smets, P. (1990). Computational aspects of the Möbius transformation. In *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence*, pages 401–416. Elsevier Science.

Koivisto, M. and Sood, K. (2004). Exact Bayesian structure discovery in Bayesian networks. *Journal of Machine Learning Research*, 5:549–573.

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

Kuipers, J. and Moffa, G. (2015). Uniform random generation of large acyclic digraphs. *Statistics and Computing*, 25(2):227–242.

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

Liskovets, V. (1975). On the number of maximal vertices of a random acyclic digraph. *Theory of Probability and its Applications*, 20:401–409.

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

Melançon, G., Dutour, I., and Bousquet-Mélou, M. (2001). Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, 10:202–207.

Melançon, G. and Philippe, F. (2004). Generating connected acyclic digraphs uniformly at random. *Information Processing Letters*, 90(4):209–213.

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.

Robinson, R. W. (1973). Counting labeled acyclic digraphs. In *New Directions in the Theory of Graphs*, pages 239–273. Academic Press, New York.

Tian, J. and He, R. (2009). Computing posterior probabilities of structural features in Bayesian networks. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 538–547. AUAI Press.

Yates, F. (1937). *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science.