Averaging of Decomposable Graphs by Dynamic Programming and Sampling

Kustaa Kangas **Teppo Niinimäki** **Mikko Koivisto**
University of Helsinki, Department of Computer Science,
Helsinki Institute for Information Technology HIIT, Finland
{jwkangas,tzniinim,mkhkoivi}@helsinki.fi

Abstract

We give algorithms for Bayesian learning of decomposable graphical models from complete data. We build on a recently proposed dynamic programming algorithm that finds optimal graphs of n nodes in $O(4^n)$ time and $O(3^n)$ space (Kangas et al., NIPS 2014), and show how it can be turned into accurate averaging algorithms. Specifically, we show that certain marginals of the posterior distribution, like the posterior probability of an edge, can be computed in $O(n^3 3^n)$ time, provided that the prior over the graphs is of an appropriate form. To overcome some limitations of the exact approach, we also give sampling schemes that—using essentially no extra space—can draw up to 3^n independent graphs from the posterior in $O(n4^n)$ time. Through importance sampling, this enables accurate Bayesian inference with a broader class of priors. Using benchmark datasets, we demonstrate the method's performance and the advantage of averaging over optimization when learning from little data.

1 INTRODUCTION

A *decomposable graphical model* represents conditional independence relations between a set of variables by an undirected graph that is *decomposable*, or equivalently, *triangulated* or *chordal*. Due to their various nice properties—in particular, convenient parameterization, straightforward parameter learning from complete data, and computationally efficient inference—decomposable models have played a central role in both methodological and applied works (Lauritzen and Spiegelhalter 1988, Dawid and Lauritzen 1993, Abel and Thomas 2011).

It is common that the modeller hesitates to fix any specific graph, and would rather like to learn the graph from data. Learning the graph has, however, proved to be computationally very challenging (Srebro 2003, Corander et al. 2013). The Bayesian approach, in particular, requires us to turn a prior into a posterior over all possible decomposable graphs. Or more practically, the interest is in drawing a representative (random) sample from the posterior or summarizing the posterior by some of its marginals. Solving these tasks by exhaustive enumeration of all graphs is feasible only up to about eight nodes. To manage with larger graphs, several Markov chain Monte Carlo (MCMC) methods have been proposed (Madigan and York 1995, Giudici and Green 1999, Tarantola 2004, Corander et al. 2006, Green and Thomas 2013). While these methods may work well in many cases, they offer essentially no guarantees concerning the accuracy of the produced estimates.

In this paper, we present exact averaging and sampling algorithms that admit Bayesian learning of decomposable graphs up to about 20 nodes. We build on the recent dynamic programming (DP) algorithm of Kangas et al. (2014). Their algorithm finds a decomposable graph that *maximizes* a given decomposable scoring function, for example, the posterior probability. For graphs with n nodes the algorithm takes $O(4^n)$ time and $O(3^n)$ space. Generally, it is straightforward to turn a DP algorithm, designed for an optimization problem, into a "corresponding" averaging or sampling algorithm (and vice versa). At first glance, our result may thus look nothing but a rephrasing of the result of Kangas et al.

However, this view turns out to be only partial. Indeed, replacing maximization by averaging opens new algorithmic opportunities, which allow us to give an asymptotically faster, $O(n^3 3^n)$-time algorithm (Sect. 3). On the other hand, the averaging viewpoint also brings new difficulties: First, it turns out that the particular DP treatment applies to Bayesian model averaging only when the prior over the graphs is of a specific form; for example, the form cannot express the uniform prior over all decomposable graphs, whereas it can express the prior that is proportional to the number of so-called rooted junction trees of the graph. Second, it appears that exact computations are feasible only for marginal posterior probabilities of small subgraphs, like of an individual edge, but not of large subgraphs, not to mention more complicated graph features. We overcome these difficulties by designing schemes that, using the computed DP tables, can efficiently generate large numbers of random samples from the posterior or its computationally convenient proxy (Sect. 4). We then feed the samples to usual Monte Carlo estimators of the quantities of interest (Sect. 5).

We demonstrate the performance of the methods using benchmark datasets (Sect. 6). First, we investigate whether averaging over graphs yields significantly better prediction results as compared to using a single maximum-a-posteriori graph. Second, we study the accuracy of the sampling based estimators for edge posterior probabilities.

Finally, we discuss straightforward ways to further enhance the presented methods, and also mention some major open questions (Sect. 7).

2 PRELIMINARIES

We review some fundamentals of decomposable models. For a more thorough treatment, see Lauritzen (1996).

2.1 DECOMPOSABILITY AND LEARNING

Consider an undirected graph G on a set of nodes $V = \{1, \ldots, n\}$. We call a subset of nodes *complete* if it induces a complete subgraph. Further, we call a complete set a *clique* if it is *maximal*, that is, not a subset of any other complete set. We denote the set of cliques of G simply by $\mathcal{C}$, assuming there is no ambiguity about the referred graph.

We call G *decomposable* if it has the *running intersection property*, that is, there is an ordering of its cliques, $C_1, \ldots, C_k$, such that for each $i = 2, \ldots, k$ we have that $S_i = (C_1 \cup \cdots \cup C_{i-1}) \cap C_i \subset C_j$ for some $j < i$. The sets S_i are called the *separators* and they form a multiset, which we denote by $\mathcal{S}$. We note that $\mathcal{S}$ is uniquely specified by G and does not depend on the ordering of the cliques.

Suppose G is decomposable and with each node $v \in V$ associate a random variable x_v. The graph G together with a joint distribution p over the variables form a *decomposable graphical model* if p factorizes as

$$p(x_V) = \frac{\prod_{C \in \mathcal{C}} p(x_C)}{\prod_{S \in \mathcal{S}} p(x_S)} \,,$$

where we write x_S for the tuple $(x_v : v \in S)$. The factorization plays a central role in probabilistic inference in a given decomposable model, and in learning a model for a fixed graph from *data*, that is, multiple records over the variables.

The factorization is central also when learning the graph. In the Bayesian approach to learning we turn a *prior* ρ over the graphs into a *posterior* π by multiplying the prior by the (marginal) *likelihood* ℓ, and dividing by the *normalizing constant* of the function $\rho\ell$, given as $Z_{\rho\ell} = \sum_G \rho(G)\ell(G)$.

The likelihood $\ell(G)$ is the probability (density) of the data, given G. It is obtained by integrating out the parameters that specify the distribution p above. Under commonly adopted parameter priors (Dawid and Lauritzen 1993), the likelihood function factorizes into a product of local marginal likelihoods, one term per clique and separator. Thus the likelihood function is an example of a decomposable function:

Definition 1 (decomposable function)**.** Let V be a finite set and let φ be a function from the decomposable graphs on V to real numbers. We say that φ is *decomposable* over V if

$$\varphi(G) = \frac{\prod_{C \in \mathcal{C}} \varphi_{\mathrm{c}}(C)}{\prod_{S \in \mathcal{S}} \varphi_{\mathrm{s}}(S)}$$

for some functions φ_{c} and φ_{s}, we call the *local components*.

Clearly the product of two decomposable functions is also decomposable. It can also be shown that any constant function is decomposable. In particular, if the prior is a decomposable function, so is the posterior.

We illustrate the notion of decomposable functions by two further examples. Here and henceforth we use the Iverson bracket $[Q]$ to denote 1 when Q is true, and 0 otherwise.

Example 1 (uniform prior)**.** Let w be a number. Let ρ be the decomposable function over V defined by

$$\rho_{\mathrm{c}}(X) = [\,|X| \le w\,] \quad \text{and} \quad \rho_{\mathrm{s}}(X) = 1 \quad \text{for } X \subseteq V\,.$$

We observe that $\rho(G) = 1$ if G contains only cliques of size at most w, and 0 otherwise. Thus the normalized function ρ/Z_ρ is the uniform distribution over the decomposable graphs on V whose cliques are of size at most w.

Example 2 (absence of an edge)**.** Let $e \subseteq V$, $|e| = 2$. Let φ^e be the decomposable function over V defined by

$$\varphi^e_{\mathrm{c}}(X) = [e \not\subseteq X] \quad \text{and} \quad \varphi^e_{\mathrm{s}}(X) = 1 \quad \text{for } X \subseteq V\,.$$

We observe that $\varphi^e(G) = 1$ if the edge e is absent in G, and 0 otherwise. Thus φ^e is the indicator function of the set of decomposable graphs that do not contain the edge e.

Note also that the indicator function for the *presence* of an edge is not decomposable.

2.2 COMPUTATIONAL TASKS

We will consider two classes of computational problems: (1) computing the *marginal* of a given function φ of decomposable graphs, defined as $Z_\varphi = \sum_G \varphi(G)$; and (2) generating random samples of decomposable graphs from a distribution that is proportional to a given function. The first class includes the important task of computing posterior expectations of graph features, in particular, marginal posterior probabilities of edges (cf. Corollary 2 in Sect. 3). In the second class the distribution of interest is usually the posterior distribution. The input in these problems is always specified by decomposable functions, like a prior and a likelihood function. Thus we may assume an efficient access to these functions through their local components.

2.3 ROOTED JUNCTION TREES

It is convenient to represent a decomposable graph as a *junction tree*. Consider an undirected graph G and a tree $\mathcal{J}$ that has the cliques of G as its vertices. We call $\mathcal{J}$ a junction tree of G if it satisfies the *junction property*, that is, for any cliques C, C' the intersection $C \cap C'$ is contained within every clique on the unique path between C and C' in $\mathcal{J}$. A graph has a junction tree if and only if it is decomposable. Importantly, the representation is in general not unique, as a graph may have multiple junction trees; we denote by $\tau(G)$ the number of junction trees of G. In contrast, for each junction tree $\mathcal{J}$ the represented graph, $G(\mathcal{J})$, is unique.

For our purposes it will be convenient to work with rooted junction trees. Specifically, we make use of the following recursive characterization of junction trees, adopted from the work of Kangas et al. (2014):

Definition 2 (recursive partition tree, RPT)**.** A *recursive partition tree* over a finite ground set V is a triplet $(C, \{R_1, \dots, R_k\}, \{\mathcal{T}_1, \dots, \mathcal{T}_k\})$ such that

1. C is a non-empty subset of V, called the *root*;
2. $\{R_1, \dots, R_k\}$ is a partition of $V \setminus C$;
3. each $\mathcal{T}_i$ is an RPT over $C \cup R_i$ rooted at C_i such that $C \cap C_i$ is a proper subset of both C and C_i.

Note that the implicit base case of the definition is when $C = V$ and there are thus no subtrees.

An RPT $\mathcal{T}$ rooted at C can be viewed as a directed tree, where $C_1, \dots, C_k$ are the children and $\mathcal{T}_1, \dots, \mathcal{T}_k$ the subtrees of C. Such a tree is a junction tree and, conversely, any junction tree can be represented as an RPT, which is unique up to the choice of the root (Kangas et al. 2014). Denoting by $\kappa(G)$ the number of cliques of a decomposable graph G, we have that G has exactly $\tau(G)\kappa(G)$ distinct RPTs.

2.4 DYNAMIC PROGRAMMING

Kangas et al. considered the problem of maximizing a given decomposable function, and gave a DP algorithm that stems from the recursive definition of RPTs. We next adapt the DP algorithm to the problem of summing up the values of a given function, that is, to compute the marginal. Due to the many-to-one relationship of RPTs and decomposable graphs, each value gets multiplied by the corresponding number of RPTs. Put otherwise, for any function φ we have

$$\sum_{\mathcal{T}} \varphi\left(G(\mathcal{T})\right) = \sum_{G} \varphi(G)\tau(G)\kappa(G)\,, \tag{1}$$

where $\mathcal{T}$ runs through all RPTs over V and G runs through all decomposable graphs on V. Since our DP treatment relies on the decomposability of the function φ, the computed sum will not be the marginal of the decomposable φ, but of the function $\varphi\tau\kappa$, that we shall call RPT-decomposable:

Definition 3 (RPT-decomposable function)**.** Let V be a finite set and let φ' be a function from the decomposable graphs on V to real numbers. We say that φ' is *RPT-decomposable* over V if $\varphi'(G) = \varphi(G)\tau(G)\kappa(G)$ for some decomposable function φ over V.

Example 3 (RPT-uniform prior)**.** Let ρ be as defined in Example 1. Then $\rho\tau\kappa$ is a RPT-decomposable function that is proportional to the uniform distribution on all RPTs over V whose cliques are of size at most w.

We are ready to present the DP algorithm for computing the sum (1) for a given a decomposable function φ. To this end, we denote by $\mathrm{RPT}(S, R)$ the set of all RPTs over $S \cup R$ rooted at a proper superset of S, and let

$$f(S, R) = \sum_{\mathcal{T} \in \mathrm{RPT}(S,R)} \varphi(G(\mathcal{T}))\,.$$

In particular, $f(\varnothing, V)$ equals the desired sum (1). Following Kangas et al. we obtain the following recurrence system:

$$f(S, R) = \sum_{S \subset C \subseteq S \cup R} \varphi_{\mathrm{c}}(C)\, g(C, R \setminus C)\,, \tag{2}$$

$$g(C, U) = \sum_{\min U \in R \subseteq U} h(C, R)\, g(C, U \setminus R)\,, \tag{3}$$

$$h(C, R) = \sum_{S \subset C} f(S, R)/\varphi_{\mathrm{s}}(S)\,, \tag{4}$$

with the base case $g(C, \varnothing) = 1$. Each recurrence is defined for all disjoint pairs of subsets of V such that C and R are non-empty. A straightforward evaluation of f, g, and h by using these recurrences takes $O(4^n)$ time and $O(3^n)$ space. These bounds and the correctness of the recurrences can be verified essentially by taking the proof of Kangas et al. and replacing maximization with summation.

The intuition here is that f considers all possible choices for the root clique C and factors in the $\varphi_{\mathrm{c}}(C)$. For each C it then invokes g, which considers partitions $\{R_1, \dots, R_k\}$ of the remaining nodes U by calling itself recursively. For each part R it also calls h, which considers possible separators S between C and the subtree in R and factors in the reciprocal of the marginals $\varphi_{\mathrm{s}}(S)$. Finally, h calls f again to consider possible root cliques of the subtree. In (3), the least element of U, denoted $\min U$, is always placed in the next part R so as not to consider different permutations of the same partition. The base case corresponds to the case where all nodes have been assigned to some R_i and there are none left to partition.

3 EXACT AVERAGING

In this section we show:

Theorem 1. *The marginal of a given RPT-decomposable function over a set of n nodes can be computed in $O(n^3 3^n)$ time and $O(n3^n)$ space.*

In effect, we show that the recurrence system (2–4) can be solved in the claimed time. Asymptotically, this is significantly faster than the $O(4^n)$ time obtained by a straightforward evaluation. We achieve the improvement by computing the exponential-size summations for each of the functions f, g, and h simultaneously for several pairs of arguments. Our algorithms for f and h apply as well to the maximization variant of the recurrence. In contrast, our algorithm for g relies crucially on subtraction (i.e., the existence of additive inverses), and thus does not apply to maximization.

Before we proceed to the proof (in Sections 3.1–3.5), let us note the following implication:

Corollary 2. *Suppose the probability distribution over decomposable graphs on a set of n nodes is proportional to a given RPT-decomposable function. Then the probability that the graph contains k specified edges can be computed in $O(2^k n^3 3^n)$ time and $O(n3^n)$ space.*

Proof. Let $e_1, \ldots, e_k$ be distinct edges on the node set $V = \{1, \ldots, n\}$. Let A_j denote the event that the graph does *not* contain the edge e_j. By the inclusion–exclusion principle, the probability that the graph contains the k edges is

$$\Pr\left[\bar{A}_1 \cap \cdots \cap \bar{A}_k\right] = \sum_{J \subseteq \{1,\ldots,k\}} (-1)^{|J|} \Pr\left[\bigcap_{j \in J} A_j\right] .$$

Now, let φ' be the given RPT-decomposable function. It remains to observe that

$$\Pr\left[\bigcap_{j \in J} A_j\right] = \frac{\sum_G \varphi'(G) \prod_{j \in J} \varphi^{e_j}(G)}{\sum_G \varphi'(G)} ,$$

where each φ^{e_j} is the decomposable function that indicates whether e_j is absent in the graph, as defined in Example 2. Thus each of the 2^k probabilities is obtained as a ratio of two marginals of RPT-decomposable functions. □

3.1 ZETA TRANSFORM AND SUBSET CONVOLUTION

Our algorithms employ the so-called *fast zeta transform*, FZT (Yates 1937, Kennes and Smets 1990, Björklund et al. 2012). The *zeta transform* of a function α from the subsets of a ground set $\{1, \ldots, n\}$ to real numbers is the set function defined by $\hat{\alpha}(Y) = \sum_{X \subseteq Y} \alpha(X)$ for each subset Y of the ground set. FZT computes the zeta transform of a given function in $O(n2^n)$ time, as follows: Let $\alpha_0 = \alpha$ and for $i = 1, \ldots, n$ let

$$\alpha_i(Y) = \alpha_{i-1}(Y) + [i \in Y] \cdot \alpha_{i-1}(Y \setminus \{i\}) .$$

It follows that $\alpha_n = \hat{\alpha}$. Note that there are two different ways to organize the computations: either compute the n steps one after another, which requires only $O(2^n)$ space; or compute all the n functions for each set Y one after another in increasing order by the size $|Y|$, which requires $O(n2^n)$ space. We will need the latter "level-wise" implementation.

Another tool we use is known as the *fast subset convolution*, FSC (Björklund et al. 2007). The *subset convolution* of two functions α and β from the subsets of a ground set $\{1, \ldots, n\}$ to real numbers is the set function defined by $(\alpha * \beta)(Y) = \sum_{X \subseteq Y} \alpha(X)\beta(Y \setminus X)$ for each subset Y of the ground set. FSC computes the subset convolution in $O(n^2 2^n)$ time and $O(n2^n)$ space. We refer to Björklund et al. (2007) for details.

3.2 COMPUTING h

Consider first the recurrence for h. For a moment, fix a set $R \subseteq V$ and define the functions h_R and f_R by

$$h_R(C) = h(C, R) \quad \text{and} \quad f_R(S) = f(S, R)/\varphi_s(S) ,$$

where C and S are subsets of $V \setminus R$. By the recurrence (4) we have that $h_R(C) = \sum_{S \subset C} f_R(S) = \hat{f}_R(C) - f_R(C)$. Using FZT we can compute h_R in $O(k2^k)$ time for each R, where $k = n - |R|$. Note that we use the level-wise implementation and evaluate the k steps of the transform for one C in turn. Thus computing h takes $O(n3^n)$ time in total. (While we here used subtraction, it is not difficult to see how FZT can be modified to avoid that, and that the result thus applies to the maximization variant as well.)

3.3 COMPUTING f

Consider then the recurrence for f. It might be tempting to try the above trick also in this case, that is, to fix either R or S, and then employ a suitable fast zeta transform variant. However, that approach fails because now the involved three sets S, R, and C have more intricate dependencies. Instead, we define the function g' by

$$g'(C, R \setminus C) = \varphi_c(C)\, g(C, R \setminus C) ,$$

for $C \subseteq R \subseteq V$. The idea is to extend FZT to pairs of disjoint sets and transform g' into f by computing the sum (2) in n steps: Let $g'_0 = g'$ and for $i = 1, \ldots, n$ let

$$g'_i(S, R) = g'_{i-1}(S, R) + [i \in R] \cdot g'_{i-1}(S \cup \{i\}, R \setminus \{i\}) .$$

It can be shown by simple induction that $g'_n(S, R) = f(S, R) + g'(S, R \setminus S)$ (see the supplement). Thus, given g', we can compute f in $O(n3^n)$ time. Note that we use the level-wise approach to compute the values $f(S, R)$ in *decreasing* order of $|S|$ and in *increasing* order of $|R|$.

3.4 COMPUTING g

Finally consider the recurrence for g. Now we fix a $C \subseteq V$ for a moment and, for convenience, write $g_C(U)$ for $g(C, U)$

and $h_C(R)$ for $h(C, R)$. We will show that the values

$$g_C(U) = \sum_{\min U \in R \subseteq U} h_C(R)\, g_C(U \setminus R)$$

can be computed for all $U \subseteq V$ of size $|U| = u$ in $O(n^2 2^n)$ time, assuming the values $h_C(R)$ and $g_C(U')$ are available for all $R, U' \subseteq V$ of sizes $|R| \leq u$ and $|U'| < u$. This then implies that computing g takes $O(n^3 3^n)$ time in total (by summing over u and C).

To compute the values $g_C(U)$, we break the computations further into n separate subtasks. For each $s \in V$, define the function g^s by

$$g^s(U) = \sum_{s \in R \subseteq U} h(R)\, g(U \setminus R)\,,$$

where $U \subseteq V$ such that $|U| = u$ and $\min U = s$. Note that each such U is of the form $\{s\} \cup X$ with $X \subseteq V \setminus \{1, \ldots, s\}$. We observe that for each s the task can be solved using FSC in $O(n^2 2^{n-s})$ time, implying a time requirement of $O(n^2 2^n)$ in total. It remains to observe that $g(U)$ is obtained as $g^{\min U}(U)$.

3.5 SPACE REQUIREMENT

The level-wise computations of h and f assume that the intermediate values of the transforms are stored in memory. This incurs a space requirement of $(n3^n)$ in total.

The computation of g, instead, uses FSC in a black-box fashion and computes a bunch of values $g(C, U)$ for a fixed C and several sets U of a fixed size u, assuming only that the values $h(C, R)$ and $g(C, U')$ are available for all R and U' such that $|R| \leq u$ and $|U'| < u$. Because the same space can be reused for different C, the "extra" space requirement is only that of FSC, $O(n2^n)$.

4 SAMPLING

We now turn to the task of sampling decomposable graphs from the posterior. Specifically, we extend the DP algorithm of Sect. 2 with a natural backtracking procedure that draws an RPT by choosing its cliques, partitions, and separators according to their conditional marginal probabilities, and returns the corresponding graph. Since each graph G has $\tau(G)\kappa(G)$ distinct RPTs, the resulting graph sample follows the RPT-decomposable distribution $\pi \propto \varphi\tau\kappa$, where φ is the decomposable function that the DP algorithm was applied to. By employing importance sampling (Sect. 5) that weights each graph sample G by $(\tau(G)\kappa(G))^{-1}$, we are able to target the distribution proportional to φ instead.

For the remainder of this section we focus on sampling from $\pi \propto \varphi\tau\kappa$. Given the DP tables for f, g, and h (for φ), the backtracking procedure to sample from π works as follows.

Algorithm: Sampling a decomposable graph

Begin by visiting $f(\varnothing, V)$ and recursively visit f, g and h as follows: In $f(S, R)$, choose the root clique C of the subtree over $S \cup R$ randomly according to its marginal distribution $\Pr(C) \propto \varphi_c(C)\, g(C, R \setminus C)$ and proceed to corresponding $g(C, R \setminus C)$. In $g(C, U)$, choose the first part R of the partition according to its marginal distribution $\Pr(R) \propto h(C, R)\, g(C, U \setminus R)$ and recursively proceed to $h(C, R)$ and $g(C, U \setminus R)$. In $h(C, R)$, choose a separator S according to its marginal probability $\Pr(S) \propto f(S, R)/\varphi_s(S)$ and proceed to $f(S, R)$. Continue the recursion until all branches terminate at a visit to $g(C, \varnothing)$. Then, from the resulting recursion tree, obtain the cliques C and return the corresponding graph.

The efficiency of this backtracking procedure depends on how much time is spent on choosing random sets C, R, and S during the visits. Consider the general problem of drawing a sample from a discrete distribution over s elements. A naive method is to pick a number r from the continuous uniform distribution on $[0, 1)$, then iterate the elements in a predefined order and select the first item for which the cumulative probability exceeds r. The method uses $O(s)$ time and $O(1)$ space. It can be shown that, if the sets C, R, and S are sampled using this naive method, sampling a single graph requires $O(2^n)$ time. (In fact, this result is proved as a special case in the next subsection.) Next we introduce a more general sampling scheme that allows us to spend less time in sampling by using more space.

4.1 TRADING TIME FOR SPACE

We can avoid spending exponential time per sample by first preprocessing the probabilities so that the sets can thereafter be chosen much faster. First, it should be noted that it is not possible to just precompute the cumulative probabilities and use, for example, binary search to find the item that corresponds to r, without increasing the asymptotic space requirement. On the other hand, if we allow additional space usage, then it is better to use the *alias method* (Vose 1991), which requires $O(s)$ additional space and $O(s)$ preprocessing time but allows us to draw samples in $O(1)$ time per sample. Moreover, it turns out we obtain a tunable tradeoff between sampling time and extra space. Specifically, we propose the following algorithm:

Subroutine: Parameterized sampling

Let $b \in \{0, ..., n\}$ be a tradeoff parameter that is chosen beforehand. As preprocessing, divide the s elements into bins of size 2^b (at most, the last bin can be smaller), and for each bin, compute and store the sum of the probabilities of its elements. This requires space that is linear in the number of bins.

Now a new sample can be drawn in two phases: first a bin is selected according to its precomputed total probability and then a term is selected from the bin according to its relative probability in the bin. We apply the alias method to the first phase, and the naive sampling to the second phase. Thus, the first phase requires $O(1)$ time per sample. The additional space requirement is linear in the number of bins. The second phase uses $O(2^b)$ time and no additional space. The total space requirement is thus $O(s/2^b)$ and time requirement is $O(s)$ for preprocessing and $O(2^b)$ per sample.

By selecting a fixed b and applying the above algorithm to each (nonterminating) $f(S, R)$, $g(C, U)$, and $h(C, R)$ we get the following theorem:

Theorem 3. *For any $b \in \{0, \dots, n\}$ we can draw T independent graphs from π in $O(4^n + T \cdot 2^b(1 + n - b))$ time and $O(4^n/2^b + 3^n)$ space.*

In order to prove the theorem, we first bound the number of visits to f, g, and h by the following lemma:

Lemma 4. *The sampling procedure makes at most n nonterminating visits to functions f, g, and h each.*

The proof of Lemma 4 is given in the supplement.

Now, by using Lemma 4 and the fact that on each visit to f, g or h, the time required to choose the set C, R or S correspondingly is at most $O(2^b)$, we get a bound $O(2^b n)$ for the time consumption per sample. However, in order to get the bound down to $O(2^b(1+n-b))$ as in Theorem 3, we need to bound the amount of work done on each visit more carefully. To this end, observe that each nonterminating visit to $f(S, R)$, $g(C, U)$, and $h(C, R)$ involves drawing a random set from a discrete distribution over $2^{|R|} - 1 < 2^{|S|+|R|}$, $2^{|U|}/2 < 2^{|C|+|U|}$, and $2^{|C|} - 1 < 2^{|C|+|R|}$ sets respectively. The following lemma bounds $|S| + |R|$, $|C| + |U|$, and $|C| + |R|$ on different steps of the backtracking.

Lemma 5. *For f, g, and h let $(S_1, R_1), \dots, (S_{d_f}, R_{d_f})$, $(C_1, U_1), \dots, (C_{d_g}, U_{d_g})$, and $(C_1, R_1), \dots, (C_{d_h}, R_{d_h})$ be all the set pairs visited during backtracking. Then there exist orderings of those set pairs such that,*

$$|S_i| + |R_i| \le n - i + 1 \quad \textit{for all } i = 1, \dots, d_f ,$$

$$|C_i| + |U_i| \le n - i + 1 \quad \textit{for all } i = 1, \dots, d_g ,$$

$$|C_i| + |R_i| \le n - i + 1 \quad \textit{for all } i = 1, \dots, d_h .$$

The proof of Lemma 5 is given in the supplement.

Now we are ready to prove the theorem presented above.

Proof of Theorem 3. Consider the space and time needed for f (respectively: g, h).

Space: For each R (respectively: U, C) of size k there are 2^{n-k} ways to choose S (respectively: C, R). The space needed by the alias method for each such set pair is less than $\max\{2^k/2^b, 1\}$. Thus the total space requirement for all set pairs of all sizes is

$$\begin{aligned}\sum_k \binom{n}{k} 2^{n-k} \max\{2^{k-b}, 1\} \;&\le\; \sum_k \binom{n}{k} (2^{n-b} + 2^{n-k}) \\ &=\; 4^n/2^b + 3^n \ .\end{aligned}$$

Time: The preprocessing requires enumeration over all terms of the sums in the recurrences for all entries of f, g, and h. Like computing f, g, and h, this consumes $O(4^n)$ time. It remains to analyze the time needed to draw a single graph sample. Let $(S_1, R_1), \dots, (S_d, R_d)$ (respectively: $(R_1, U_1), \dots, (R_d, U_d)$, $(C_1, R_1), \dots, (C_d, R_d)$) be the d visited set pairs for f (respectively: g, h). In each visit the algorithm first chooses a bin in constant time and then chooses one of its elements in time linear in its size but at most 2^b. Therefore, the backtracking requires at most time

$$\sum_{i=1}^{d} \min\{2^b, 2^{k_i}\} \ ,$$

where $k_i = |R_i|$ (respectively: $k_i = |U_i|$, $k_i = |C_i|$). By Lemma 5, there exists an ordering of the set pairs such that $k_i \le n - i + 1$. By Lemma 4, $d \le n$. We get

$$\begin{aligned}\sum_{j=1}^{n} \min\{2^b, 2^j\} \;&\le\; \sum_{j=b+1}^{n} 2^b + \sum_{j=1}^{b} 2^j \\ &\le\; (n-b)2^b + 2^{b+1} \\ &=\; (2 + n - b)2^b \ .\end{aligned}$$

The claim thus follows. □

Theorem 3 has, for example, the following corollaries. Setting $b = 0$ allows us to draw each sample in linear time but using $O(4^n)$ extra space, while setting $b = n$ effectively yields the naive method. From an asymptotical viewpoint, it makes sense to set $b \le n \log_2(4/3) \simeq 0.42n$, since for larger b the 3^n term dominates the space requirement. Since the DP phase requires $O(4^n)$ time, we can in a sense draw $O(4^n/2^{n \log_2(4/3)} n) = O(2^n/n)$ samples for "free."

4.2 ADAPTIVE SAMPLING

Usually most of the probability mass is concentrated on a small set of graphs. Thus, when sampling graphs, some of the indexing set pairs for f, g, and h are visited rarely if at all. The additional space that is used by the alias method for such set pairs may outweigh the gain in speed. As an alternative, we propose the following approach that periodically draws and caches multiple samples at once on those indices that are visited more often.

Subroutine: Adaptive sampling

On the first visit to any $f(S, R)$, $g(C, U)$, or $h(C, R)$, draw and consume one set from the corresponding discrete distribution using the naive method. On any subsequent visit: If there are no cached samples left from the previous visits, then use the alias method to draw twice as many sets as the last time on the same index, consume one and cache the rest. Otherwise, consume one set from the cache.

The following lemma characterizes the space consumption of the graph sampler that uses the above adaptive sampling subroutine.

Lemma 6. *After T sampled graphs, adaptive sampling consumes at most $O(nT)$ extra space.*

Proof. A cache of size s is constructed when the corresponding set pair is visited for the $(s+1)$th time. By Lemma 4 the total number of visits to all set pairs is at most nT. The claim then trivially follows. $\square$

Furthermore, we get the following special case bounds for space and time consumption:

Theorem 7. *We can draw 3^n independent samples from π in $O(n4^n)$ time and $O(n3^n)$ space.*

Proof. The space complexity follows from Lemma 6.

Consider then the time used on visits to f.

Without rebuilding the caches, the time per sample is $O(n)$, or $O(n3^n)$ in total. In order to analyze the time needed for cache rebuilds, consider an arbitrary disjoint set pair (S, R) and let $k = |R|$.

Let t be the number of visits to the set pair in question. Then its cache is rebuilt $\lfloor \log_2(t) + 1 \rfloor$ times. As rebuilding the cache the ith time requires $O(2^k + 2^i)$ time (the first term comes from constructing the distribution and initializing the alias method, the second term comes from drawing the sets), in total this requires time

$$\sum_{i=1}^{\lfloor \log_2(t)+1 \rfloor} O(2^k + 2^i) = O((\log(t) + 1)2^k) + O(t) \, .$$

To get the total time used to rebuild caches, we must sum these for all set pairs (S, R). Since each backtracking visits at most n set pairs (Lemma 4), the sum over the last term $O(t)$ results in $O(n3^n)$. It remains to analyze the first term.

For $k = 1, \ldots, n$, let L_k consist of all the set pairs (S, R) such that $|R| = k$. For a fixed k, there are $|L_k| = \binom{n}{k} 2^{n-k}$ such pairs, and each may be visited at most once per sample, that is, at most 3^n times in total. Thus, the sum of the first term over all pairs in L_k amounts to $O\left(|L_k|(\log(3^n) + 1)2^k\right) = O\left(n\binom{n}{k}2^n\right)$. Summing over k yields the claimed running time bound.

The time used on visits to g and h can be bounded analogously (selecting $k = |U|$ and $k = |C|$, respectively). $\square$

5 MONTE CARLO ESTIMATION

We have observed in Sections 2 and 3 that the posterior probabilities of some graph properties can be computed exactly in $O(4^n)$ time, and asymptotically even faster in $O(n^3 3^n)$ time. However, we had to assume (i) that the properties can be expressed by decomposable functions (e.g., in terms of forbidden cliques), and (ii) that the prior, and hence the posterior, is RPT-decomposable, ruling out the natural uniform prior.

In this section, we employ Monte Carlo methods to relax these restrictive assumptions and still obtain good approximations. Throughout the section, we consider an arbitrary function ψ from decomposable graphs to $\{0, 1\}$. Our interest is in estimating the quantity

$$Z_{\pi\psi} = \sum_G \pi(G)\psi(G) \, ,$$

where π is the posterior distribution.

5.1 RPT-DECOMPOSABLE PRIOR

Consider first relaxing only the first assumption (i). Since we keep the assumption that the prior is RPT-decomposable, we can draw T independent graphs $G_1, \ldots, G_T$ from the posterior π, using the methods described in Sect. 4. The estimate

$$\widehat{Z}_{\pi\psi} = \frac{1}{T}\sum_{i=1}^{T} \psi(G_i)$$

is unbiased and, by the law of large numbers, it concentrates around $Z_{\pi\psi}$ as T grows.

5.2 DECOMPOSABLE PRIOR

Consider then relaxing also the second assumption (ii). For convenience, assume however that the prior is decomposable, for example, the uniform distribution over all decomposable graphs on the node set V. Now, we can draw T independent graphs $G_1, \ldots, G_T$ from a distribution proportional to $\pi\tau\kappa$, using the methods described in Sect. 4. In this case, we need to correct the deviance of the sampling distribution from the posterior. We do this by using the self-normalized importance sampling estimate

$$\widetilde{Z}_{\pi\psi} = \frac{\sum_{i=1}^{T} w_i \psi(G_i)}{\sum_{i=1}^{T} w_i} \, , \quad w_i = \frac{1}{\tau(G_i)\kappa(G_i)} \, .$$

This estimate concentrates around $Z_{\pi\psi}$ as T grows.

To make this method practical, we need an efficient way to count the number of junction trees $\tau(G_i)$ and the number of cliques $\kappa(G_i)$ of a given decomposable graph G_i. The latter problem is easy, as the cliques are readily available in the RPT (or junction tree) representation. For counting junction

Table 1: Benchmark datasets on n variables and m records.

Dataset	n	m
Asia	8	10000
Bridges	12	108
Flare	13	1066
House-votes	17	435

Table 2: The running time of our algorithm on the benchmark datasets. We measure in seconds the time spent on the dynamic programming phase (DP) as well as sampling of 10^5, 10^6, and 10^7 graphs. These include the time spent finding the number of junction trees per sample.

Dataset	DP	10^5	10^6	10^7
Asia	0.018	0.32	3.2	33
Bridges	5.6	0.66	5.5	54
Flare	24	1.5	8.8	77
House-votes	7497	16	40	167

trees, a method described by Thomas and Green (2009) has, to our knowledge, the best worst case guarantees, running in time $O(n^2)$. It also starts by finding a single junction tree of the graph, which in our case is already given.

6 EXPERIMENTS

We report experimental results on the proposed sampling and estimation methods, using benchmark datasets (Table 1). *Asia* is sampled from a network defined by Lauritzen and Spiegelhalter (1988) and others are from the UCI repository (Bache and Lichman 2013). For all datasets we use the Dirichlet–multinomial model with the equivalent sample size parameter 1 (Dawid and Lauritzen 1993, Heckerman et al. 1995). Our C++ implementation[1] uses the straightforward $O(4^n)$ time dynamic programming (Sect. 2) and the adaptive variant of the sampling phase (Sect. 4). The running times are detailed in Table 2.

6.1 PREDICTION

We first study how well averaging over graphs using our sampling method performs on a prediction task, as compared to using a single maximum-a-posteriori graph. Specifically, we split each dataset into a fixed test set and a training set. From the training data we then learn both a maximum-a-posteriori graph and the full posterior given a uniform prior over decomposable graphs and measure how the probability of the test data given the training data behaves under these models as the size of the training set varies.

The maximum-a-posteriori graph is obtained using the method of Kangas et al. (2014). The full posterior model is approximated by sampling 2^n graphs, where n is the number of variables in the dataset, and then measuring the average probability of the test data over the sampled graphs (weighting each graph by the number of its RPTs).

Intuitively, the full posterior should in general yield better results, as the maximum-a-posteriori graph has a tendency to overfit the training data, especially for a small training set. Our results (Fig. 1) conform to this expectation. We observe that the probability of the test data under the full posterior model tends to be significantly higher for small training sets and the gap diminishes as the size of the training set grows.

6.2 EDGE POSTERIOR PROBABILITIES

In the second set of experiments we apply the Monte Carlo methods described in Sect. 5 to estimating the posterior probabilities of individual edges under the uniform prior over decomposable models.

For each dataset, we use the method from Sect. 5.2 to estimate the posterior probability of each edge. We compute the *estimation error*, i.e., the difference between an edge's estimated probability and our best available estimate for the probability. Ideally, we would like the best available estimate to be the true posterior probability of the edge. For *Asia* we are able to obtain the true probability by enumerating all decomposable graphs up to 8 nodes. As this task comes infeasible for larger n, for other datasets we instead compare to the best estimate given by our method after drawing one million samples. Such "self-comparison" is still useful for studying the rate of convergence.

To quantify convergence, we measure how the maximum estimation error over edges develops as we draw more samples (Fig. 2). We also study the distribution of the error in a more refined way by counting the number of edges whose error exceeds 10^{-k} for $k \in \{1, \dots, 4\}$ (Fig. 3). We observe that the rate of convergence remains steady for each dataset. In all cases we reach a maximum error of about 0.01 or less after 10^5 samples and a lot sooner for smaller datasets.

7 CONCLUDING REMARKS

We have presented algorithms for Bayesian learning of moderate-size decomposable graphical models. Unlike the recent related algorithm that finds a single optimal model (Kangas et al. 2014), our algorithm enables more principled treatment of the posterior uncertainty. Unlike the brute-force approach, our algorithm scales well beyond 8 variables, up to about 20 variables. Unlike the popular MCMC methods, our algorithm enjoys accuracy guarantees (see also below). We believe our algorithms can be valuable for solving actual data-analysis instances, for testing the performance of other (approximate) methods, and as a building block for developing new methods that scale to larger instances.

[1]Our implementation is available at `www.cs.helsinki.fi/u/jwkangas/junctor`.

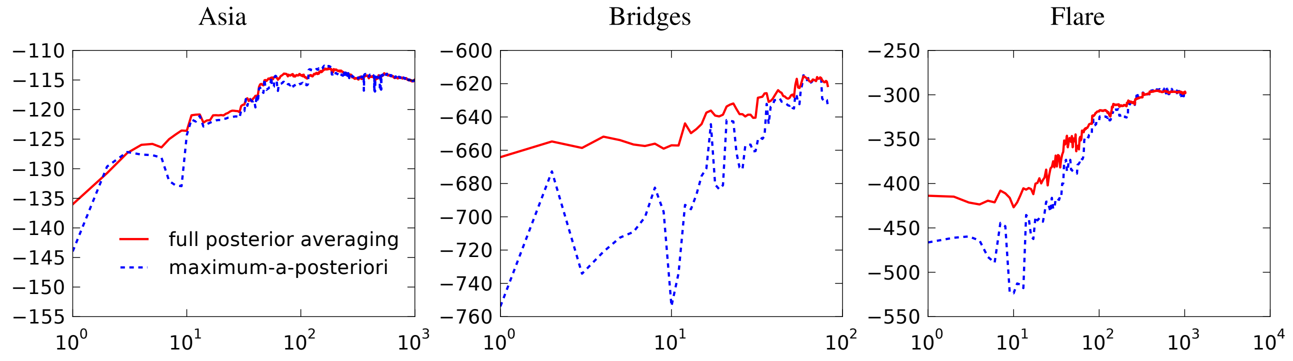

Figure 1: The (log unnormalized) probability of the test data given the training data (y-axis) as an average under sampled graphs and under the maximum-a-posteriori model for various sizes of the training set (x-axis).

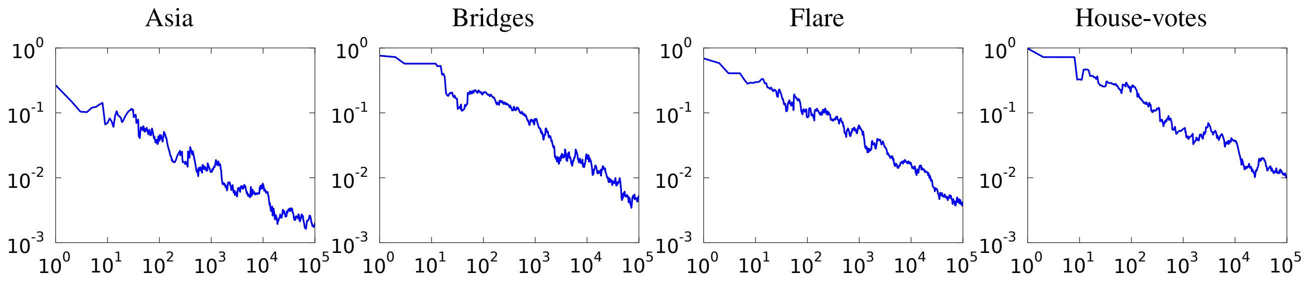

Figure 2: The maximum error of the estimate over all edges (y-axis) as the number of samples (x-axis) grows.

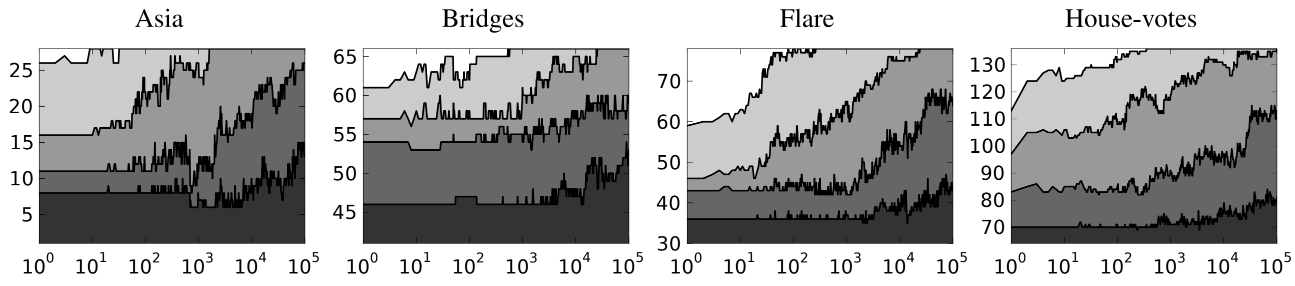

Figure 3: The number of edges (y-axis) with an estimation error less than 10^{-k} as the number of samples (x-axis) grows. The curves represents $k = 1, \ldots, 4$ from top to down.

While this work has touched several aspects of accurate Bayesian learning of decomposable graphs, it also leaves many questions for future work. Some are rather straightforward extension and implementation issues: From a practical point of view, the most important one is to make the algorithms accommodate a user-specified upper bound on the clique sizes, and thereby expedite computations and reduce memory requirements. Another issue is to tune the recursive sampling schemes so as to fully exploit the (typical) low entropy of the distributions. Conceptually, the biggest shortcoming in our current importance sampling implementation is the lack of controllable approximation guarantees. However, we believe this gap can be closed in an efficient and practical way by using the so-called optimal Monte Carlo algorithms (Cheng 2001, Dagum et al. 2000), as the sampled graphs tend to have relatively few junction trees.

The main open research questions are: Can the asymptotically faster algorithm be implemented to run fast also in practice? Are there significantly faster and, particularly, more space-efficient algorithms for Bayesian learning of decomposable graphs, with good accuracy guarantees?

Acknowledgements

The authors thank the anonymous reviewers for valuable suggestions to improve the presentation. This work was supported in part by the Academy of Finland, Grant 276864 “Supple Exponential Algorithms” (M.K.).

References

H. Abel and A. Thomas. Accuracy and computational efficiency of a graphical modeling approach to linkage disequilibrium estimation. *Statistical Applications in Genetics and Molecular Biology*, 10:1–15, 2011.

K. Bache and M. Lichman. UCI machine learning repository, 2013.

A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74. ACM, 2007.

A. Björklund, M. Koivisto, T. Husfeldt, J. Nederlof, P. Kaski, and P. Parviainen. Fast zeta transforms for lattices with few irreducibles. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1436–1444. SIAM, 2012.

J. Cheng. Sampling algorithms for estimating the mean of bounded random variables. *Computational Statistics*, 16: 1–23, 2001.

J. Corander, M. Gyllenberg, and T. Koski. Bayesian model learning based on a parallel MCMC strategy. *Statistics and Computing*, 16(4):355–362, 2006.

J. Corander, T. Janhunen, J. Rintanen, H. Nyman, and J. Pensar. Learning chordal Markov networks by constraint satisfaction. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 1349–1357. Curran Associates, Inc., 2013.

P. Dagum, R. M. Karp, M. Luby, and S. M. Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing*, 29(5):1484–1496, 2000.

A. P. Dawid and S. L. Lauritzen. Hyper Markov laws in the statistical analysis of decomposable graphical models. *The Annals of Statistics*, 21(3):1272–1317, 1993.

P. Giudici and P. J. Green. Decomposable graphical Gaussian model determination. *Biometrika*, 86(4):785–801, 1999.

P. J. Green and A. Thomas. Sampling decomposable graphs using a Markov chain on junction trees. *Biometrika*, 100 (1):91–110, 2013.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

K. Kangas, M. Koivisto, and T. Niinimäki. Learning chordal Markov networks by dynamic programming. In *Advances in Neural Information Processing Systems 27 (NIPS)*, pages 2357–2365. Curran Associates, Inc., 2014.

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

S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

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

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

N. Srebro. Maximum likelihood bounded tree-width Markov networks. *Artificial Intelligence*, 143(1):123–138, 2003.

C. Tarantola. MCMC model determination for discrete graphical models. *Statistical Modelling*, 4(1):39–61, 2004.

A. Thomas and P. J. Green. Enumerating the junction trees of a decomposable graph. *Journal of Computational and Graphical Statistics*, 18:930–940, 2009.

M. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17:972–975, 1991.

F. Yates. *The design and analysis of factorial experiments*. Imperial Bureau of Soil Science. Harpenden, 1937.