Exact Learning of Bounded Tree-width Bayesian Networks

Janne H. Korhonen
University of Helsinki
Dept. of Computer Science and HIIT
Helsinki, Finland

Pekka Parviainen
Royal Institute of Technology
CSC and Scilifelab
Stockholm, Sweden

Abstract

Inference in Bayesian networks is known to be NP-hard, but if the network has bounded tree-width, then inference becomes tractable. Not surprisingly, learning networks that closely match the given data and have a bounded tree-width has recently attracted some attention. In this paper we aim to lay groundwork for future research on the topic by studying the exact complexity of this problem. We give the first non-trivial exact algorithm for the NP-hard problem of finding an optimal Bayesian network of tree-width at most w, with running time $3^n n^{w+O(1)}$, and provide an implementation of this algorithm. Additionally, we propose a variant of Bayesian network learning with "super-structures", and show that finding a Bayesian network consistent with a given super-structure is fixed-parameter tractable in the tree-width of the super-structure.

1 INTRODUCTION

1.1 Bayesian network learning

Bayesian networks are used widely to represent joint probability distributions. Typically, the first step in using a Bayesian network to model some problem is *learning* the network from the input data. That is, we have to learn a directed acyclic graph (DAG) and the parameters associated with each variable, so that the model describes the original data "well". Learning the parameters given a structure is an easy task, so in the recent years research has mostly focused on learning the structure. One of the main approaches to structure learning is so-called score-based methods (Cooper and Herskovits, 1992; Heckerman et al., 1995), where the idea is to assign each possible structure a score based on how well it fits the data and try to find a structure that maximises the score.

Appearing in Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS) 2013, Scottsdale, AZ, USA. Volume 31 of JMLR: W&CP 31. Copyright 2013 by the authors.

In this paper, we study the Bayesian structure learning as a combinatorial problem using an abstract score-based framework. In this framework, we are given a node set N of size n and for each node $v \in N$ and each parent set $S \subseteq N \setminus \{v\}$ a local score $f_v(S)$. The goal is to find a DAG A that maximises the sum

$$f(A) = \sum_{v \in N} f_v(A_v) \,,$$

where A_v is the parent set of v, *i.e.*, the set of nodes u such that there is an arc from u to v in A. This problem is NP-hard (Chickering, 1996; Chickering et al., 2004), the best known exact algorithm being a Bellman–Held–Karp style dynamic programming that runs in time $2^n n^{O(1)}$ (Silander and Myllymäki, 2006).

1.2 Learning with bounded tree-width

Once the network has been learned, we want to use it to compute conditional probabilities of some sets of variables given some other sets of variables. This *inference problem* in Bayesian networks is also NP-hard (Cooper, 1990). However, if the network (or more precisely its structure) has low *tree-width*, exact inference is tractable even for large networks. Thus, learning models of bounded tree-width enables us to limit the time required for inference. Specifically, we have a trade-off between the fit of the network and the speed of inference, since if the "true" network has high tree-width, bounding the tree-width can lead to under-fitting.

More formally, given local scores f_v as before and a constant w, we want to find a DAG A that maximises the score $f(A)$ among the DAGs of tree-width at most w. Here the tree-width of a DAG is defined as the tree-width of its moralised graph (Elidan and Gould, 2008); the *moralised graph* of a DAG A is an undirected graph that includes an edge $\{u, v\} \in E$ for every arc $uv \in A$ and an edge $\{u, w\} \in E$ for every pair of arcs $uv \in A$ and $wv \in A$. Defined this way, the tree-width of a network matches its inference complexity[1].

While there have been some studies on learning undirected models with bounded tree-width using approximation algorithms (Karger and Srebro, 2001; Srebro, 2001), heuristics (Bach and Jordan, 2002), and PAC-learning (Chechetka and Guestrin, 2008), the corresponding problem for Bayesian networks remains poorly understood. The only result of this vein we are aware of is a heuristic algorithm for learning bounded tree-width Bayesian network structures by Elidan and Gould (2008). Our main motivation for the work presented in this paper is to fill this gap and lay groundwork for future investigations of the topic. Specifically, we aim to establish basic theoretical results for learning bounded tree-width Bayesian network structures, especially in the sense of *exact* algorithmics.

Unfortunately, learning Bayesian network structures remains difficult when the tree-width is bounded. While learning an optimal *tree*, i.e., a Bayesian network with tree-width 1, can be done in polynomial time (Chow and Liu, 1968), a straightforward reduction from a corresponding result for Markov networks shows that finding an optimal Bayesian network of tree-width at most w is NP-hard for any *fixed* $w \geq 2$; see Section 2.3.

1.3 Learning in exponential time

Since learning bounded tree-width Bayesian networks is NP-hard, the natural question from the perspective of exact algorithmics is to study exponential-time algorithms for the problem. As our main result, we obtain a single-exponential time algorithm for bounded tree-width Bayesian structure learning.

Theorem 1. *Given a node set N of size n, an integer w and scoring functions f_v for each node $v \in N$, we can find a DAG A with tree-width at most w maximising score $f(A) = \sum_{v \in N} f_v(A_v)$ in time $3^n n^{w+O(1)}$ and space $2^n n^{w+O(1)}$.*

The proof of Theorem 1 is given in Section 5.

Somewhat disappointingly we are not able to match the $2^n n^{O(1)}$ algorithm for unrestricted Bayesian network structure learning. Indeed, it seems to us that the added restriction of the bounded tree-width makes the problem more challenging.

On the practical side, we have implemented the algorithm of Theorem 1, and it works well for small n and w. We also experimented with using this implementation to find small bounded tree-width networks for real-world data; see Section 6.

Although the exponentiality hinders the application of the algorithm for all but a small number of nodes, we argue that having even an exponential exact algorithm for the problem is essential for further investigations of the topic. Principally, it provides a baseline against which approximation algorithms and heuristics can be tested, and it may also prove to be useful as a component of such algorithms. Furthermore, being able to generate examples of optimal bounded tree-width networks enables explorative studies of their properties.

We also note that in the recent years there has been a lot of interest in exponential time algorithms for learning the structure of a Bayesian network (Ott and Miyano, 2003; Singh and Moore, 2005; Silander and Myllymäki, 2006) and related tasks, like computing posterior probabilities of structural features (Koivisto and Sood, 2004; Koivisto, 2006; Tian and He, 2009; Kang et al., 2010; Parviainen and Koivisto, 2011). Most of the algorithms run in $2^n n^{O(1)}$ time but some of them have running time $3^n n^{O(1)}$ which matches our algorithm. In line with our experiments, other algorithms with running time $3^n n^{O(1)}$ been implemented and tested successfully with networks of up to 20 nodes (Tian and He, 2009; Kang et al., 2010; Parviainen and Koivisto, 2011).

1.4 Learning with super-structures

The dynamic programming algorithm of Theorem 1 implicitly contains a subroutine that, given an undirected graph G, finds an optimal DAG whose moralised graph is a subgraph of G. Indeed, this problem is fixed-parameter tractable with regard to the tree-width of the graph G, as formalised in the following theorem, whose proof we give in Section 4.

Theorem 2. *For any fixed w, given an n-vertex graph $G = (N, E)$ of tree-width at most w and scoring functions f_v for each node $v \in N$, we can find a DAG A whose moralised graph is a subgraph of G maximising the score in time and space $O(n)$.*

Specifically, the running time of our algorithm is $O\big((w+1)! \cdot w \cdot 3^w \cdot n\big)$ if we are given a suitable tree-decomposition of G. As it is usual with algorithms based on tree-decompositions, the bottle-neck is the construction of a tree-decomposition from G; see Section 2.1.

This observation is related to the super-structure approach for learning Bayesian networks, presented by Perrier et al. (2008), where we are given an undirected graph G, called the super-structure, and the goal is to

[1]To avoid confusion, we point out that this definition differs from the definition of tree-width for directed graphs given by Johnson et al. (2001).

find the highest-scoring DAG such that there is, for each arc in the DAG, a corresponding undirected edge in G, i.e. the *skeleton* of the DAG is a subgraph of the super-structure. Recently Ordyniak and Szeider (2010) have shown that in this setting, if both the tree-width and the maximum degree of the super-structure are bounded by constants, an optimal DAG can be found in linear time. However, the tree-width of the super-structure alone does not bound the running time of the algorithm or the tree-width of the resulting DAG, and Ordyniak and Szeider in fact show that learning optimal Bayesian network with given super-structure is $W[1]$-hard when the complexity parameter is the tree-width of the super-structure. Intuitively, the reason for this is that the tree-width of a DAG is defined to be the tree-width of its moralised graph, and moralising can introduce edges that are not present in the super-structure.

2 PRELIMINARIES

2.1 Tree-width

For an undirected graph $G = (V, E)$, we use the convention that E is a set of two-element subsets of V. We write $\{u, v\}$ for an edge between nodes u and v. We also denote $n = |V|$.

A *tree-decomposition* of an undirected graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$, where $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$ is a collection of subsets of V and T is a tree on $\{1, 2, \ldots, m\}$, such that

1. $\bigcup_{i=1}^{m} X_i = V$,
2. for all edges $\{u, v\} \in E$ there exist i with $u \in X_i$ and $v \in X_i$, and
3. for all i, j and k, if j is on the (unique) path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *width* of a tree-decomposition $(\mathcal{X}, T)$ is defined as $\max_i |X_i| - 1$. The *tree-width* of an undirected graph G is the minimum width over all possible tree-decompositions of G. In a sense, the tree-width of a graph describes how close the graph is to a tree; graphs of tree-width 1 coincide with trees. For a fixed w and graph G with tree-width w, a tree-decomposition of width w can be found in time $O(n)$ (Bodlaender, 1996).

A *nice tree-decomposition* of a graph $G = (V, E)$ is a tree-decomposition $(\mathcal{X}, T)$ along with a fixed root node r for T such that each node $i \in \{1, 2, \ldots, m\}$ is either

1. a *leaf* with no children and $|X_i| = 1$,
2. a *forget* node that has one child j and $X_i = X_j \setminus \{v\}$ for some $v \in X_j$,
3. a *introduce* node that has one child j and $X_i = X_j \cup \{v\}$ for some $v \notin X_j$, or
4. a *join* node that has two children j and k and $X_i = X_j = X_k$.

For fixed w, if we are given a tree-decomposition of G with width w, we can construct a nice tree-decomposition of width w and a linear number of nodes in time $O(n)$ (Kloks, 1994). Thus, if we are given a graph of tree-width w, we can also obtain a nice tree-decomposition of width w and a linear number of nodes in time $O(n)$, since some tree-decomposition of width w can be found in linear time, as noted above.

We will in particular need the following basic property of the tree-decompositions.

Lemma 3 (Separation Property)**.** *Let $G = (V, E)$ be a graph with tree-decomposition $(\mathcal{X}, T)$, and let i, j and k be nodes in T such that j is on the path from i to k. Then there is no edge $\{u, v\} \in E$ with $v \in X_i \setminus X_j$ and $u \in X_k \setminus X_j$.*

Finally, we give a well-known alternate characterisation of tree-width. The family of *k-trees* is defined inductively as follows.

1. A $(k + 1)$-clique is a k-tree.
2. If $G = (V, E)$ is a k-tree and $C \subseteq V$ is a k-clique, then graph obtained by adding a new vertex v and an edge uv for each $u \in C$ is a k-tree.

The k-trees are maximal graphs with tree-width k, that is, a graph has tree-width k if and only if it is a subgraph of some k-tree; see e.g. van Leeuwen (1990).

2.2 Bayesian Networks

Aside from the definitions given in Sections 1.2 and 1.4, we will use the following conventions when discussing Bayesian networks and the structure learning problem.

A directed graph is a pair (N, A), where N is the node set and $A \subseteq N \times N$ is the arc set. We write uv for arc $(u, v) \in A$. When there is no ambiguity about the node set, we identify a directed graph by its arc set. Throughout this paper, we denote $n = |N|$.

A node u is said to be a *parent* of node v if the arc set contains an arc from u to v, that is, $uv \in A$. If u is a parent of v, then v is a *child* of u. We denote the set of the parents of v in A by A_v.

As a consequence of the definition of the tree-width of of a DAG (see Section 1.2), we have that if the tree-width of a DAG is w, then the in-degree must be bounded by w, as $\{v\} \cup A_v$ is a clique in the moralised graph M, and a graph with k-clique has tree-width at least $k-1$. The reverse does not hold, and a graph with maximum in-degree Δ can have tree-width larger than Δ.

In the structure learning problem, we are given the local scores $f_v(S)$ for each node $v \in N$ and each parent set $S \subseteq N \setminus \{v\}$ as input. The output is a DAG that has maximal score. As noted above, parent sets S of size more than w are not eligible when we want the DAG to have tree-width at most w. Thus, we assume that the input consist only of the scores for parent sets of size at most w and has size $O\left(n\binom{n}{w}\right)$. For structure learning with super-structures, we may further assume that we are given scores only for parent sets that are compatible with the super-structure, in which case the input has size $O(n2^w)$. We will not, however, consider the representation of the input in more detail, and in the rest of the paper we will assume that we can access scores $f_v(S)$ in constant time. This does not have significant effect on the analysis.

Finally, we define functions $\hat{f}_v$ by $\hat{f}_v(S) = \max_{T \subseteq S} f_v(T)$. That is, $\hat{f}_v(S)$ is the highest local score when the parents of node v are chosen from S. For any set X, the values $\hat{f}_v(S)$ for *all* sets $S \subseteq X$ can be computed from the values of f_v in $O\big(|X|2^{|X|}\big)$ time and $O\big(2^{|X|}\big)$ space using dynamic programming (Ott and Miyano, 2003).

2.3 Hardness

Srebro (2000) has shown that the problem of finding a subgraph of an input graph G with tree-width at most w and maximum number of edges is NP-hard for any fixed $w \geq 2$. The NP-hardness of learning bounded tree-width Bayesian network structures follows by a straightforward reduction.

Theorem 4. *Finding an optimal Bayesian network structure with tree-width at most w under a given scoring function is NP-hard for any fixed $w \geq 2$.*

Proof. Let $G = (V, E)$ be an undirected graph. Let $N = V \cup E$, and define a score function on N by setting $f_e(\{v, u\}) = 1$ for each edge $e = \{v, u\} \in E$, and let $f_v(S) = 0$ for any other node $v \in N$ and potential parent set $S \subseteq N \setminus \{v\}$. This transformation can be computed in polynomial time.

Now we note that there is a subgraph (V, F) of G with $|F| = m$ and tree-width at most w if and only if there is a DAG D on N with $f(D) = m$ and tree-width at most w; furthermore, if we are given one, we can compute the other in polynomial time. Since finding the maximum bounded tree-width subgraph is known to be NP-hard, the claim follows. □

3 DECOMPOSING DAGS

In this section, we establish results that will be used to prove the correctness of the algorithms we give later on. The intuitive idea is that if (N, A) is a DAG of low tree-width, then there is a small set $X \subseteq N$ whose removal will split A into two or more connected components. We can exploit this property by finding optimal DAGs that can act as these separate components, and then "glue" these DAGs together at X. We now proceed to formalise these ideas.

Let N be a set of nodes and let $X \subseteq N$ with $|X| = k$. For a permutation $\sigma = \sigma_1\sigma_2\ldots\sigma_k$ of X and a set $S \subseteq X$, we say that a DAG A is a *(σ, S)-DAG on N* if the node set of A is N, it holds that A is *compatible* with σ, that is, $A \cup \{\sigma_p\sigma_{p+1} \colon p = 1, 2, \ldots, k-1\}$ is acyclic, and for each $v \in X \setminus S$ we have that $A_v = \emptyset$. For a (σ, S)-DAG A on N, we define the *S-score* of A as $f_S(A) = \sum_{v \in S \cup (N \setminus X)} f_v(A_v)$. That is, the nodes in X that are required to have empty parent sets do not contribute to the score.

In the following, we assume that N is some node set, and X, N_1 and N_2 are subsets of N such that $N_1 \cup N_2 = N$ and $N_1 \cap N_2 = X$. Furthermore, we assume that σ is a permutation of X and $S \subseteq X$.

Lemma 5. *Let $Z \subseteq S$. If A is a (σ, Z)-DAG on N_1 and B is a $(\sigma, S \setminus Z)$-DAG on N_2, then $A \cup B$ is a (σ, S)-DAG on N. Furthermore, we have*

$$f_S(A \cup B) = f_Z(A) + f_{S \setminus Z}(B)\,.$$

Proof. The claim follows almost directly from the definitions; the only non-trivial step to verify is that $A \cup B$ is in fact acyclic. To see that $A \cup B$ is acyclic, assume that there is a directed cycle C in $A \cup B$. Since both A and B are acyclic, there must be a node σ_i on cycle C. But since both A and C are compatible with σ, each maximal segment of C that consists only of edges in A or only of edges in B goes from a node σ_j to a node σ_ℓ for $j < \ell$, and thus the cycle cannot return to σ_i. □

For a (σ, S)-DAG A on N, we say that *a decomposition of A over N_1 and N_2* is a pair (B, C), where B is a (σ, Z)-DAG on N_1 and C is a $(\sigma, S \setminus Z)$-DAG on N_2 such that $A = B \cup C$ and $Z \subseteq S$. Note that if A has such a decomposition (B, C), then by Lemma 5 we have $f_S(A) = f_Z(B) + f_{S \setminus Z}(C)$.

Lemma 6. *Suppose that A is an (σ, S)-DAG on N and suppose there are no arcs in A between $N_1 \setminus X$ and $N_2 \setminus X$, and no $v \in N$, $u \in N_1 \setminus X$ and $w \in N_2 \setminus X$ such that $uv \in A$ and $wv \in A$. Then there is a decomposition of A over N_1 and N_2.*

Proof. Let $Z = \{v \in S \colon A_v \subseteq N_1\}$. Then the DAGs $B = \{uv \colon v \in Z \cup (N_1 \setminus X)\}$ and $C =$

$\{uv\colon v \in (S \setminus Z) \cup (N_2 \setminus X)\}$ clearly have the desired properties. □

Lemma 7. *Suppose that $\mathcal{A}$ is a family of (σ, S)-DAGs on N and that for each $Z \subseteq S$, we have that $\mathcal{B}_Z$ is a family of (σ, Z)-DAGs on N_1 and $\mathcal{C}_Z$ a family of (σ, Z)-DAGs on N_2. If each $A \in \mathcal{A}$ has a decomposition (B, C) over N_1 and N_2 such that $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S\setminus Z}$ for some $Z \subseteq S$, and for each $Z \subseteq S$ and DAGs $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S\setminus Z}$ it holds that $B \cup C \in \mathcal{A}$, then*

$$\max_{A\in\mathcal{A}} f_S(A) = \max_{Z\subseteq S}\Big(\max_{B\in\mathcal{B}_Z} f_Z(B) + \max_{C\in\mathcal{C}_{S\setminus Z}} f_{S\setminus Z}(C)\Big)\,.$$

Proof. Fix $A \in \mathcal{A}$. Since A decomposes into $B \in \mathcal{B}_Z$ and $C \in \mathcal{C}_{S\setminus Z}$ for some $Z \subseteq S$, we have

$$\begin{aligned} f_S(A) &= f_Z(B) + f_{S\setminus Z}(C) \\ &\le \max_{B\in\mathcal{B}_Z} f_Z(B) + \max_{C\in\mathcal{C}_{S\setminus Z}} f_{S\setminus Z}(C) \\ &\le \max_{Z\subseteq S}\Big(\max_{B\in\mathcal{B}_Z} f_Z(B) + \max_{C\in\mathcal{C}_{S\setminus Z}} f_{S\setminus Z}(C)\Big)\,. \end{aligned}$$

On the other hand, since $B \cup C \in \mathcal{A}$ for all $B \in \mathcal{B}_Z$ and $C \in C_{S\setminus Z}$, there is a DAG in $\mathcal{A}$ with S-score

$$\max_{Z\subseteq S}\Big(\max_{B\in\mathcal{B}_Z} f_Z(B) + \max_{C\in\mathcal{C}_{S\setminus Z}} f_{S\setminus Z}(C)\Big)\,.$$ □

4 LEARNING WITH SUPER-STRUCTURES

We prove Theorem 2 first. The proof of this theorem will act as a preliminary for the proof of Theorem 1 in the next section.

To prove Theorem 2, we use dynamic programming on the tree-decomposition of the underlying super-structure. We will assume that parent sets that are not compatible with the super-structure graph G have score of $-\infty$ and will thus not be picked by the algorithm. That is, if for $v \in V$ we have that $S \subseteq V \setminus \{v\}$ contains a node u such that $\{u, v\} \notin E$, or nodes u and s such that $\{u, s\} \notin E$, then $f_v(S) = -\infty$.

Let $G = (V, E)$ be the super-structure graph and let $(\mathcal{X}, T)$ be a nice tree-decomposition of G with root r and $\mathcal{X} = \{X_1, X_2, \ldots, X_m\}$. For $1 \le i \le m$, we denote by V_i the union of bags below X_i in the tree.

Intuitively, our algorithm will proceed by computing for each node i in T an optimal (σ, S)-DAG on V_i for each permutation σ of X_i and $S \subseteq X_i$. We will show that these can be computed by starting from the leaves of T and proceeding upwards in the tree. Finally, in the root r, we will have the optimal (σ, X_i)-DAG on V for each permutation of X_i; taking the one with the best score gives us the desired optimal Bayesian network.

To formalise the intuition given above, let $i \in \{1, 2, \ldots, m\}$ and let $k = |X_i|$. For a permutation $\sigma = \sigma_1\sigma_2\ldots\sigma_k$ of X_i and $S \subseteq X_i$, we define

$$g_i(\sigma, S) = \max_A f_S(A)\,, \tag{1}$$

where A ranges over (σ, S)-DAGs on V_i such that the moralised graph of A is a subgraph of $G[V_i]$. It follows immediately from this definition that the best DAG on V has score $\max_\sigma g_r(\sigma, X_r)$, where σ ranges over the permutations of the root bag X_r. Furthermore, we note that it suffices to show how these scores can be computed, as the optimal DAG can then be recovered using standard techniques; see e.g., Silander and Myllymäki (2006).

The values $g_i(\sigma, X_i)$ can be computed using dynamic programming on the tree-decomposition, starting from the leaf nodes and going up in the tree. There are four cases to consider, depending on the type of node X_i.

Leaf: $X_i = \{v\}$ for $v \in N$. Then we have $g_i(v, \emptyset) = 0$ and $g_i(v, \{v\}) = f_v(\emptyset)$.

Forget: Node i has a child j and $X_i = X_j \setminus \{v\}$ for $v \in X_j$. Now $V_i = V_j$, and directly by definition we have that

$$g_i(\sigma, S) = \max_{\eta\tau=\sigma} g_j(\eta v\tau, S \cup \{v\}) \tag{2}$$

for all permutations σ of X_i and $S \subseteq X_i$. Computing (2) directly for all σ and S takes $O(k2^k k!)$ time.

Introduce: Node i has a child j and $X_i = X_j \cup \{v\}$ for $v \notin V_j$. First, we compute values $\hat{f}_u(S)$ for $u \in X_i$ and $S \subseteq X_i \setminus \{u\}$, which takes $O(k2^k)$ time as noted in Section 2.2. Now suppose that $\sigma = \eta v\tau$ is a permutation of X_i and $S \subseteq X_i$. Denote by $P_{\sigma,u}$ the set of elements of X_i that appear before u in σ. Then we have that

$$g_i(\sigma, S) = \max_{Z\subseteq S\setminus\{v\}}\Big(g_j(\eta\tau, Z) + \sum_{u\in S\setminus Z} \hat{f}_u(P_{\sigma,u})\Big)\,. \tag{3}$$

To verify that (3) is correct, consider any (σ, S)-DAG A on V_i. Since by Lemma 3 there are no edges $\{u, v\} \in E$ for $u \in V_i \setminus X_i$, Lemma 6 implies that A interpreted as a $(\eta\tau, S \setminus \{v\})$-DAG has a decomposition (B, C) over $V_i \setminus \{v\}$ and X_i, where B is a $(\eta\tau, Z \setminus \{v\})$-DAG on $V_i \setminus \{v\}$ and C is a $(\sigma, S \setminus Z)$-DAG on X_i for some $Z \subseteq S$. Furthermore, the moralised graphs of both B and C are subgraphs of G. Finally, we note that the maximum score for a (σ, Z)-DAG on X_i is $\sum_{u\in Z} \hat{f}_u(P_{\sigma,u})$. The correctness of (3) now follows from Lemma 7.

Evaluating (3) for all σ and S can be done in time $O(k3^k k!)$.

Join: Node i has children j and ℓ, and $X_i = X_j = X_\ell$. By Lemma 3, there are no edges between $V_j \setminus X_i$ and

$V_\ell \setminus X_i$ in G. Thus any (σ, S)-DAG A on V_i has a decomposition (B, C) over $V_j \setminus X_i$ and $V_\ell \setminus X_i$. Since moralised graphs of both B and C are subgraphs of G, Lemma 7 implies that

$$g_i(\sigma, S) = \max_{Z \subseteq S} \bigl(g_j(\sigma, Z) + g_\ell(\sigma, S \setminus Z)\bigr) . \qquad (4)$$

Evaluating (4) for all σ and S takes $O(3^k k!)$ time.

Summing the running times over all nodes in T, we obtain the following.

Theorem 8. *Given a graph G, a nice tree-decomposition $(\mathcal{X}, T)$ of G, and scoring functions f_v for each node v, we can find a DAG A whose moralised graph is a subgraph of G maximising the score in time $O\bigl((w+1)! \cdot w \cdot 3^w \cdot n\bigr)$, where w is the tree-width of G.*

As noted in Section 2.1, a nice tree-decomposition of the super-structure graph G can be obtained from G in $O(n)$ time. Thus, we obtain Theorem 2 as a corollary.

5 EXACT LEARNING WITH BOUNDED TREE-WIDTH

We will now proceed to prove Theorem 1 by giving a dynamic programming algorithm for the problem. This algorithm is based on the same ideas as the super-structure algorithm in Section 4, but here we perform dynamic programming over all possible tree-decompositions of width w. In the following, let w be a fixed tree-width bound.

As noted in Section 2.1, each graph of tree-width w is a subgraph of a w-tree. It follows that each graph $G = (V, E)$ of tree-width w has a rooted tree-decomposition $(\mathcal{X}, T)$ such that each bag X_i has size $w + 1$, and for adjacent i and j we have that $|X_i \cap X_j| = w$. By applying an obvious transformation to $(\mathcal{X}, T)$, we have that G also has a rooted tree-decomposition $(\mathcal{Y}, Q)$ such that each bag has size $w + 1$ and each node i in Q is either

1. a *leaf* with no children,
2. a *swap* node that has one child j such that $Y_i = (Y_j \setminus \{u\}) \cup \{v\}$ for some $u \in Y_j$ and $v \notin Y_j$, or
3. a *join* node that has two children j and ℓ such that $Y_i = Y_j = Y_\ell$.

Furthermore, by the construction we can assume that for a join node i with children j and ℓ, both V_j and V_ℓ contain vertices not in X_i. We will call a tree-decomposition satisfying these conditions *fat*. Thus, each graph G of tree-width w has a fat tree-decomposition of width w.

Let now N be a node set and let $X \subseteq N$. For a permutation σ of X and $S \subseteq X$, we say that a (σ, S)-DAG A on N is *rooted* if A has a fat tree-decomposition with root r and $X_r = X$. Furthermore, we say that A is *join-rooted* or *swap-rooted* if there is a fat tree-decomposition where the root node is of the corresponding type.

Now for $X \subseteq N$ with $|X| = w + 1$, a permutation σ of X, and sets $S \subseteq X$ and $M \supseteq X$, we want to compute

$$g(\sigma, S, M) = \max_A f_S(A) ,$$

where A ranges over rooted (σ, S)-DAGs on M with tree-width at most w. Computing these values is sufficient for finding an optimal DAG of tree-width w, as the optimal DAG is rooted at some $X \subseteq N$ with $|X| = w + 1$, thus has score $\max_{X,\sigma} g(\sigma, X, N)$, where X ranges over $(w + 1)$-subsets of N and σ ranges over permutations of X.

We will now show that these values can be computed using dynamic programming, starting from sets $M \subseteq N$ with $|M| = w + 1$. For any set M with $|M| = w + 1$ and a permutation σ of M, we note that a (σ, S)-DAG A on M has a fat tree-decomposition whose root r is a leaf node with $X_r = M$. Thus, any (σ, S)-DAG on M is rooted, and we have that

$$g(\sigma, S, M) = \sum_{u \in S} \hat{f}_u(P_{\sigma,u}) ,$$

as σ completely specifies the order of nodes in any (σ, S)-DAG on M.

On the other hand, if $M \subseteq N$ with $|M| > w + 1$, then the optimal rooted (σ, S)-DAG on M can be either join-rooted or swap-rooted. Therefore, we compute values

$$J(\sigma, S, M) = \max_B f_S(B) ,$$

where B ranges over join-rooted (σ, S)-DAGs on M with tree-width at most w, and

$$K(\sigma, S, M) = \max_C f_S(C) ,$$

where C ranges over swap-rooted (σ, S)-DAGs on M with tree-width at most w. Then we have that

$$g(\sigma, S, M) = \max\bigl\{K(\sigma, S, M), J(\sigma, S, M)\bigr\} .$$

The one special case is the sets M with $|M| = w+2$, as then there cannot be a join-rooted (σ, S)-DAG on M. Thus, for M with $|M| = w + 2$, we have $g(\sigma, S, M) = K(\sigma, S, M)$.

Join. First, we show how values $J(\sigma, S, M)$ can be computed. In the following, let M_1 and M_2 be sets such that $M_1 \cup M_2 = M$ and $M_1 \cap M_2 = X$. Furthermore, assume that $M_1 \neq X$ and $M_2 \neq X$.

Lemma 9. *If A is a rooted (σ, Z)-DAG on M_1 and B is a rooted $(\sigma, S \setminus Z)$-DAG on M_2, then $A \cup B$ is a join-rooted (σ, S)-DAG on M. Moreover, if A and B have tree-width at most w, so does $A \cup B$.*

Proof. The claim follows from Lemma 5 and from the observation that we can obtain the desired tree-decomposition for $A \cup B$ by adding the tree-decompositions of A and B as the children of a new root node r with $X_r = X$. □

Lemma 10. *If A is a join-rooted (σ, S)-DAG on M, then there are sets M_1 and M_2 such that $M_1 \cup M_2 = M$, $M_1 \cap M_2 = X$, $M_1 \neq X$, $M_2 \neq X$ and A has a decomposition (B, C) over M_1 and M_2 such that both B and C are rooted at X. Moreover, if A has tree-width at most w, so does B and C.*

Proof. Let $(\mathcal{X}, T)$ be a tree-decomposition of A with root r such that $X_r = X$ and r is a join node with children i and j. Let $M_1 = V_i$ and $M_2 = V_j$. Now by Lemma 3 and Lemma 6, A has decomposition (B, C) over M_1 and M_2. Noticing that the subtree of $(\mathcal{X}, T)$ rooted at i is a tree-decomposition of B and the subtree of $(\mathcal{X}, T)$ rooted at j is a tree-decomposition of C completes the proof, as both of these have width w and root bag X. □

For fixed M_1 and M_2, Lemma 9 implies that we can apply Lemma 7 similarly as in the join case of the super-structure algorithm, obtaining that the S-score of the best join-rooted (σ, S)-DAG that decomposes over M_1 and M_2 is

$$h(M_1, M_2) = \max_{Z \subseteq S}\big(g(\sigma, Z, M_1) + g(\sigma, S \setminus Z, M_2)\big).$$

As the optimal join-rooted (σ, S)-DAG on M decomposes over some M_1 and M_2 by Lemma 10, we have that

$$J(\sigma, S, M) = \max_{\substack{M_1 \cap M_2 = X \\ M_1 \cup M_2 = M \\ M_1, M_2 \neq X}} h(M_1, M_2)\,. \tag{5}$$

Evaluating (5) directly for fixed σ, S and M can be done in time

$$O\big(n \cdot 2^{|M \setminus X|} \cdot 2^{|S|}\big) = O\big(n \cdot 2^{|M| + |S| - (w+1)}\big)\,.$$

Swap. We now show how values $K(\sigma, S, M)$ can be computed. The following lemmas are analogous to Lemmas 9 and 10, and we omit their proofs.

Lemma 11. *Let $Y \subseteq M$ with $|Y| = w$, and $u \in M \setminus Y$ and $v \notin M$. Furthermore, let $\sigma = \eta v \tau$ be a permutation of $Y \cup \{v\}$ and $\gamma = \zeta u \rho$ be a permutation of $Y \cup \{u\}$ such that $\eta\tau = \zeta\rho$. If A is a (σ, S_1)-DAG on $Y \cup \{v\}$ and B is a rooted (γ, S_2)-DAG on M, then $A \cup B$ is a swap-rooted $(\sigma, S_1 \cup S_2)$-DAG on $M \cup \{v\}$. If B has tree-width at most w, then so does $A \cup B$.*

Lemma 12. *Let A be a swap-rooted (σ, S)-DAG on M. Then there are nodes $v \in X$ and $u \in M \setminus X$ such that, when we let $\sigma = \eta v \tau$ and $Y = (X \setminus \{v\}) \cup \{u\}$, there is a permutation $\gamma = \zeta u \rho$ of Y with $\zeta\rho = \eta\tau$, a (σ, S_1)-DAG B on X and a rooted (γ, S_2)-DAG C on $M \setminus \{u\}$ such that $A = B \cup C$. Furthermore, the tree-width of C is at most the tree-width of A.*

For $Y \subseteq M$ with $|Y| = w$ and a permutation γ of Y, we first compute an auxiliary function F defined by

$$F(\gamma, Z, M) = \max_{A} f_Z(A)\,,$$

where A ranges over rooted (σ, S)-DAGs on M such that σ is a permutation of a set $X \subseteq M$, for some $v \in M \setminus Y$ we have $X = Y \cup \{v\}$ and $S = Z \cup \{v\}$, and $\sigma = \eta v \tau$ with $\eta\tau = \gamma$. It follows directly from the definition that we can evaluate F as

$$F(\gamma, Z, M) = \max_{v \in M \setminus Y} \max_{\eta\tau = \gamma} g\big(\eta v \tau, Z \cup \{v\}, M\big)\,. \tag{6}$$

For a fixed $v \in X$, applying Lemmas 11 and 7 in a similar fashion as in the introduce case of the superstructure algorithm, we have that the maximum score of a swap-rooted (σ, S)-DAG on M with tree-width at most w such that the new node in the root bag is v is

$$\kappa(v) = \max_{Z \subseteq S \setminus \{v\}}\Big(F(\sigma, Z, M \setminus \{v\}) + \sum_{u \in S \setminus Z} \hat{f}_u(P_{\sigma,u})\Big)\,.$$

It then follows from Lemma 12 that the maximum score can be obtained by optimising over v, that is, we have

$$K(\sigma, S, M) = \max_{v \in X} \kappa(v)\,. \tag{7}$$

Finally, we note that evaluating (6) for fixed γ, Z and M can be done in time $O\big(w^2\big)$, and when the required values of F have been evaluated beforehand, (7) can be evaluated in time $O\big(2^{|S|} w\big)$.

The total number of tuples (σ, S, M) for all $X \subseteq N$ is $(w+1)!\binom{n}{w+1}2^n$. By summing the running times over all these tuples and estimating $\binom{n}{w+1} \leq n^{w+1}/(w+1)!$, we have that the total running time of our exact algorithm is $3^n n^{w+O(1)}$. Furthermore, we note that we need to store all values of g during the dynamic programming, meaning that the total space requirement of the algorithm is $2^n n^{w+O(1)}$. Thus, we have proven Theorem 1.

Remark 13. It is possible to recover a tree-decomposition for width w for the optimal DAG without extra computational cost in an obvious way.

Remark 14. By omitting the join step from the algorithm described in this section we can obtain a $2^n n^{w+O(1)}$ time algorithm for finding networks of bounded *path-width*.

6 EXPERIMENTS

To complement our theoretical results, we constructed a proof-of-concept implementation of the dynamic programming algorithm of Theorem 1 and tested it on real-world data sets[2]. In this section, we will discuss the performance of our implementation and provide examples of bounded tree-width networks on real-world data sets.

The implementation was made in Python, using cython compiler[3] to compile the most computationally demanding parts of the code to C, and the experiments were run under Linux on blade servers with 2.53-GHz processors and 32 GB of memory. We tested our implementation on two datasets, Adult (15 variables, 32,561 samples) and Housing (14 variables, 506 samples), downloaded from the UCI machine learning repository (Frank and Asuncion, 2010). We discretised all variables into binary variables, and as the local scores, we used BDeu scores with equivalent sample size 1.

Finding optimal networks of tree-width at most 2 with our algorithm took 13,086 and 3,220 seconds for Adult and Housing respectively; the reported times are user times measured using `time` and they include the dynamic programming itself and the recovery of the optimal network structure, but exclude the time needed for computing the local scores. We also benchmarked the implementation on various variable subsets of the aforementioned datasets with $w = 1$, $w = 2$, and $w = 3$, and the results were in line with theoretical bounds of Theorem 1. For $n = 14$ and $w = 3$, our implementation ran into problems with memory requirements, but this is mostly caused by an inefficient choice of data structure for storing dynamic programming values indexed by triples (σ, S, M), and these limits should be circumventable by a good choice of data structures and some careful algorithm engineering.

An example of optimal bounded tree-width networks for Housing found is shown in Figure 1. The network with unbounded tree-width is quite complex, with tree-width at least 6, so the networks with tree-width 1 and 2 are rough approximations. Indeed, the optimal network has score -3080, while the scores for bounded tree-width networks are -3295 for $w = 2$ and -3479 for $w = 1$. The optimal network with tree-width 2 has 23 arcs, meaning that is relatively dense, as a tree-width 2 network on 14 nodes can have at most 25 arcs. The most connected node is NOX, with 9 neighbours, in contrast to the optimal unbounded network, where NOX has only 5 neighbours. This hints that the more complex structure may allow representing dependencies indirectly. Overall, however, we do not feel that these examples suggest any hitherto unknown features of bounded tree-width networks as a model class. A more thorough study is warranted in the future.

Acknowledgements

We thank Mikko Koivisto for useful comments. The research was supported in part by the Helsinki Doctoral Programme in Computer Science - Advanced Computing and Intelligent Systems (J.K.) and by the Finnish Doctoral Programme in Computational Sciences (P.P.).

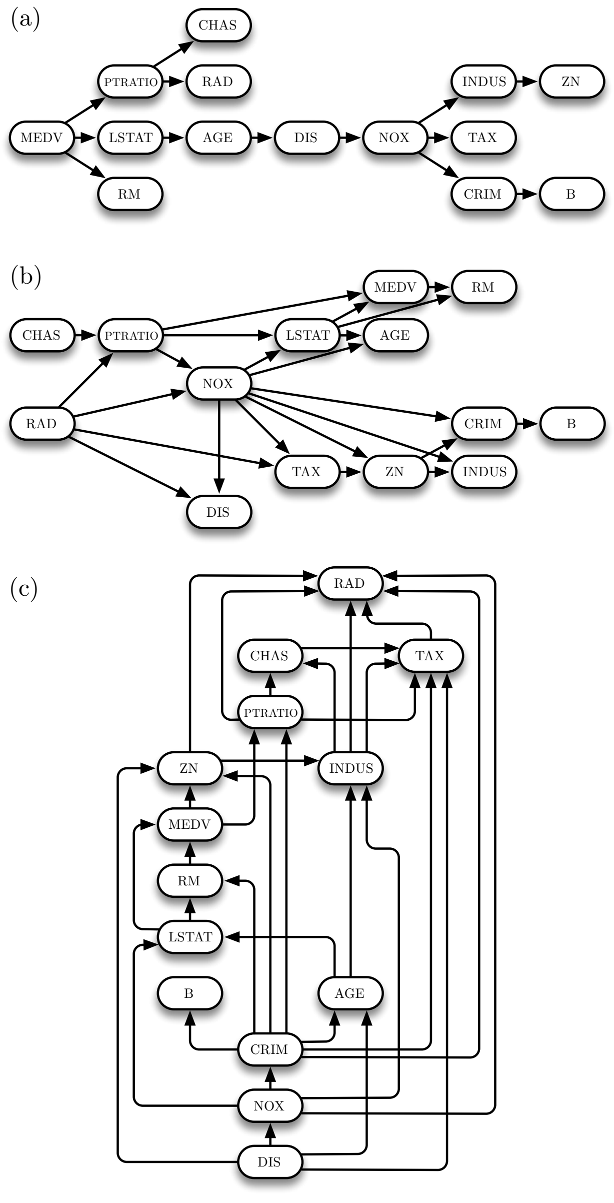

Figure 1: An optimal network for Housing for tree-width bound (a) $w = 1$, (b) $w = 2$, and (c) unbounded tree-width.

[2]The implementation is available at `http://www.cs.helsinki.fi/u/jazkorho/aistats-2013/`.

[3]`http://www.cython.org`

References

F. R. Bach and M. I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems 14 (NIPS)*. MIT Press, 2002.

H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25:1305–1317, 1996.

A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 273–280. MIT Press, 2008.

D. M. Chickering. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, 1996.

D. M. Chickering, D. Heckerman, and C. Meek. Large-sample learning of Bayesian networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3): 462–467, 1968.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

G. Elidan and S. Gould. Learning bounded treewidth Bayesian networks. *Journal of Machine Learning Research*, 9:2699–2731, 2008.

A. Frank and A. Asuncion. UCI machine learning repository, 2010.

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

T. Johnson, N. Robertson, P.D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

E. Y. Kang, I. Shpitser, and E. Eskin. Respecting Markov equivalence in computing posterior probabilities of causal graphical features. In *24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1175–1180. AAAI Press, 2010.

D. Karger and N. Srebro. Learning Markov networks: Maximum bounded tree-width graphs. In *12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 392–401, 2001.

T. Kloks. *Treewidth: computations and approximations*. Springer, 1994.

M. Koivisto. Advances in exact Bayesian structure discovery in Bayesian networks. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 241–248. AUAI Press, 2006.

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

S. Ordyniak and S. Szeider. Algorithms and complexity results for exact Bayesian structure learning. In *26th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 8–11, 2010.

S. Ott and S. Miyano. Finding optimal gene networks using biological constraints. *Genome Informatics*, 14:124–133, 2003.

P. Parviainen and M. Koivisto. Ancestor relations in the presence of unobserved variables. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 581–596. Springer, 2011.

E. Perrier, S. Imoto, and S. Miyano. Finding optimal Bayesian network given a super-structure. *Journal of Machine Learning Research*, 9:2251–2286, 2008.

T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *22nd Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 445–452. AUAI Press, 2006.

A. P. Singh and A. W. Moore. Finding optimal Bayesian networks by dynamic programming. Technical Report CMU-CALD-05-106, Carnegie Mellon University, June 2005.

N. Srebro. Maximum likelihood Markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology, 2000.

N. Srebro. Maximum likelihood bounded tree-width Markov networks. In *17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 504–511. AUAI Press, 2001.

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

J. van Leeuwen. Graph algorithms. In *Handbook of theoretical computer science, Vol. A*, pages 525–631. Elsevier, 1990.