Journal of Machine Learning Research 9 (2008) 2699-2731 Submitted 5/08; Published 12/08

Learning Bounded Treewidth Bayesian Networks

Gal Elidan GALEL@HUJI.AC.IL
Department of Statistics
Hebrew University
Jerusalem, 91905, Israel

Stephen Gould SGOULD@STANFORD.EDU
Department of Electrical Engineering
Stanford University
Stanford, CA 94305, USA

Editor: David Maxwell Chickering

Abstract

With the increased availability of data for complex domains, it is desirable to learn Bayesian network structures that are sufficiently expressive for generalization while at the same time allow for tractable inference. While the method of thin junction trees can, in principle, be used for this purpose, its fully greedy nature makes it prone to overfitting, particularly when data is scarce. In this work we present a novel method for learning Bayesian networks of bounded treewidth that employs global structure modifications and that is polynomial both in the size of the graph and the treewidth bound. At the heart of our method is a dynamic triangulation that we update in a way that facilitates the addition of chain structures that increase the bound on the model's treewidth by at most one. We demonstrate the effectiveness of our "treewidth-friendly" method on several real-life data sets and show that it is superior to the greedy approach as soon as the bound on the treewidth is nontrivial. Importantly, we also show that by making use of global operators, we are able to achieve better generalization even when learning Bayesian networks of unbounded treewidth.

Keywords: Bayesian networks, structure learning, model selection, bounded treewidth

1. Introduction

Recent years have seen a surge of readily available data for complex and varied domains. Accordingly, increased attention has been directed towards the automatic learning of large scale probabilistic graphical models (Pearl, 1988), and in particular to the learning of the graph structure of a Bayesian network. With the goal of making predictions or providing probabilistic explanations, it is desirable to learn models that generalize well and at the same time have low inference complexity or a small treewidth (Robertson and Seymour, 1987).

Chow and Liu (1968) showed that the optimal Markov or Bayesian network can be learned efficiently when the underlying structure of the network is constrained to be a tree. Learning the structure of general Bayesian networks, however, is computationally difficult (Dagum and Luby, 1993), as is the learning of simpler structures such as poly-trees (Dasgupta, 1999) or even unconstrained chains (Meek, 2001). Several works try to generalize the work of Chow and Liu (1968) either by making assumptions about the generating distribution (e.g., Narasimhan and Bilmes, 2003; Abbeel et al., 2006), by searching for a local maxima of a mixture of trees model (Meila and Jordan, 2000), or by providing an approximate method that is polynomial in the size of the graph but

exponential in the treewidth bound (e.g., Karger and Srebro, 2001; Chechetka and Guestrin, 2008). In the context of general Bayesian networks, Bach and Jordan (2002) propose a local greedy approach that upper bounds the treewidth of the model at each step. Because evaluating the bound on the treewidth of a graph is super-exponential in the treewidth (Bodlaender, 1996), their approach relies on heuristic techniques for producing tree-decompositions (clique trees) of the model at hand, and uses that decomposition as an upper bound on the true treewidth of the model. This approach, like standard structure search, does not provide guarantees on the performance of the model, but is appealing in its ability to efficiently learn Bayesian networks with an arbitrary treewidth bound.

While tree-decomposition heuristics such as the one employed by Bach and Jordan (2002) are efficient and useful on average, there are two concerns when using such a heuristic in a fully greedy manner. First, even the best of heuristics exhibits some variance in the treewidth estimate (see, for example, Koster et al., 2001) and thus a single edge modification can result in a jump in the treewidth estimate despite the fact that adding a single edge to the network can increase the true treewidth by at most one. More importantly, most structure learning scores (e.g., BIC, MDL, BDe, BGe) tend to learn spurious edges that result in overfitting when the number of samples is relatively small, a phenomenon that is made worse by a fully greedy approach. Intuitively, to generalize well, we want to learn bounded treewidth Bayesian networks where structure modifications are globally beneficial (contribute to the score in many regions of the network).

In this work we propose a novel approach for efficiently learning Bayesian networks of bounded treewidth that addresses these concerns. At the heart of our method is the idea of dynamically updating a valid moralized triangulation of our model in a particular way, and using that triangulation to upper bound the model's treewidth. Briefly, we use a novel triangulation procedure that is treewidth-friendly: the treewidth of the triangulated graph is guaranteed to increase by at most one when an edge is added to the Bayesian network. Building on the single edge triangulation, we are also able to characterize sets of edges that *jointly* increase the treewidth of the triangulation by at most one. We make use of this characterization of treewidth-friendly edge sets in a dynamic programming approach that learns the optimal treewidth-friendly chain with respect to a node ordering. Finally, we learn a bounded treewidth Bayesian network by iteratively augmenting the model with such chains.

Importantly, instead of local edge modifications, our method progresses by making use of chain structure operators that are more globally beneficial, leading to greater robustness and improving our ability to generalize. At the same time, we are able to *guarantee* that the bound on the model's treewidth grows by at most one at each iteration. Thus, our method resembles the global nature of the method of Chow and Liu (1968) more closely than the thin junction tree approach of Bach and Jordan (2002), while being applicable in practice to any desired treewidth.

We evaluate our method on several challenging real-life data sets and show that our method is able to learn richer models that generalize better on test data than a greedy variant for a range of treewidth bounds. Importantly, we show that even when models with unbounded treewidth are learned, by employing global structure modification operators, we are better able to cope with the problem of local maxima in the search and learn models that generalize better.

The rest of the paper is organized as follows. After briefly discussing background material in Section 2, we provide a high-level overview of our approach in Section 3. In Section 4 we present our treewidth-friendly triangulation procedure in detail, followed by a multiple edge update discussion in Section 5. In Section 6 we show how to learn a treewidth-friendly chain given a node ordering and in Section 7 we propose a practical node ordering that is motivated by the properties of

our triangulation procedure. We evaluate the merits of our method in Section 8 and conclude with a discussion in Section 9.

2. Background

In this section we provide a basic review of Bayesian Networks as well as introduce the graph theoretic concepts of treewidth-decompositions and treewidth.

2.1 Bayesian Networks

Consider a finite set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of random variables. A *Bayesian network* (Pearl, 1988) is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathcal{X}$. Formally, a Bayesian network over $\mathcal{X}$ is a pair $B = \langle \mathcal{G}, \Theta \rangle$. The first component, $\mathcal{G} = (V, E)$, is a directed acyclic graph whose vertices V correspond to the random variables in $\mathcal{X}$. The edges E in the graph represent direct dependencies between the variables. The graph $\mathcal{G}$ represents independence properties that are assumed to hold in the underlying distribution: each X_i is independent of its non-descendants given its parents $\mathbf{Pa}_i \subset \mathcal{X}$ denoted by $(X_i \perp \mathit{NonDescendants}_i \mid \mathbf{Pa}_i)$. The second component, Θ, represents the set of parameters that quantify the network. Each node is annotated with a *conditional probability distribution* $P(X_i \mid \mathbf{Pa}_i)$, representing the conditional probability of the node X_i given its parents in $\mathcal{G}$, defined by the parameters $\Theta_{X_i|\mathbf{Pa}_i}$. A Bayesian network defines a unique joint probability distribution over $\mathcal{X}$ given by

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}_i).$$

A *topological ordering* O_T of variables with respect to a Bayesian network structure is an ordering where each variable appears before all of its descendants in the network.

Given a Bayesian network model, we are interested in the task of probabilistic inference, or evaluating queries of the form $P_B(Y \mid Z)$ where Y and Z are arbitrary subsets of $\mathcal{X}$. This task is, in general, NP-hard (Cooper, 1990), except when $\mathcal{G}$ is tree structured. The actual complexity of inference in a Bayesian network (whether by variable elimination, by belief propagation in a clique tree, or by cut-set conditioning on the graph) is proportional to its *treewidth* (Robertson and Seymour, 1987) which, roughly speaking, measures how closely the network resembles a tree (see Section 2.2 for more details).

Given a network structure $\mathcal{G}$, the problem of learning a Bayesian network can be stated as follows: given a training set $\mathcal{D} = \{x[1], \ldots, x[M]\}$ of instances of $X \subseteq \mathcal{X}$, we want to learn parameters for the network. In the *Maximum Likelihood* setting we want to find the parameter values θ that maximize the log-likelihood function

$$\log P(\mathcal{D} \mid \mathcal{G}, \theta) = \sum_{m} \log P(x[m] \mid \mathcal{G}, \theta).$$

This function can be equivalently (up to a multiplicative constant) written as $\boldsymbol{E}_{\hat{P}}[\log P(X \mid \mathcal{G}, \theta)]$ where $\hat{P}$ is the empirical distribution in $\mathcal{D}$. When all instances in $\mathcal{D}$ are complete (that is, each training instance assigns values to all of the variables), estimating the *maximum likelihood* parameters can be done efficiently using a closed form solution for many choices of conditional probability distributions (for more details see Heckerman, 1998).

Learning the structure of a network poses additional challenges as the number of possible structures is super-exponential in the number of variables and the task is, in general, NP-hard (Chickering, 1996; Dasgupta, 1999; Meek, 2001). In practice, structure learning is typically done using a local search procedure, which examines local structure changes that are easily evaluated (add, delete or reverse an edge). This search is usually guided by a scoring function such as the MDL principle based score (Lam and Bacchus, 1994) or the *Bayesian score* (BDe) (Heckerman et al., 1995). Both scores penalize the likelihood of the data to limit the model complexity. An important characteristic of these scoring functions is that when the data instances are complete the score is *decomposable*. More precisely, a decomposable score can be rewritten as the sum

$$Score(\mathcal{G}:\mathcal{D}) = \sum_i \text{FamScore}_{X_i}(\mathbf{Pa}_i \ : \ \mathcal{D}).$$

where $\text{FamScore}_{X_i}(\mathbf{Pa}_i \ : \ \mathcal{D})$ is the *local* contribution of X_i to the total network score. This term depends only on values of X_i and $\mathbf{Pa}_{X_i}$ in the training instances.

Chow and Liu (1968) showed that maximum likelihood trees can be learned efficiently via a maximum spanning tree whose edge weights correspond to the empirical information between the two variables corresponding to the edge's endpoints. Their result can be easily generalized for any decomposable score.

2.2 Tree-Decompositions and Treewidth

The notions of tree-decompositions (or clique trees) and treewidth were introduced by Robertson and Seymour (1987).[1]

Definition 2.1: A tree-decomposition of an undirected graph $\mathcal{H} = (V,E)$ is a pair $(\{C_i\}_{i\in\mathcal{T}}, \mathcal{T})$ with $\{C_i\}_{i\in\mathcal{T}}$ a family of subsets of V, one for each node of $\mathcal{T}$, and $\mathcal{T}$ a tree such that

- $\bigcup_{i\in\mathcal{T}} C_i = V$.
- for all edges $(v,w) \in E$ there exists an $i \in \mathcal{T}$ with $v \in C_i$ and $w \in C_i$.
- for all $i,j,k \in \mathcal{T}$: if j is on the (unique) path from i to k in $\mathcal{T}$, then $C_i \cap C_k \subseteq C_j$.

∎

The treewidth of a tree-decomposition $(\{C_i\}_{i\in\mathcal{T}}, \mathcal{T})$ is defined to be $\max_{i\in\mathcal{T}} |C_i| - 1$. The treewidth $TW(\mathcal{H})$ of an undirected graph $\mathcal{H}$ is the minimum treewidth over all possible tree-decompositions of $\mathcal{H}$. An equivalent notion of treewidth can be phrased in terms of a graph that is a triangulation of $\mathcal{H}$.

Definition 2.2: An induced path $\mathcal{P} = p_1 — p_2 \ldots p_L$ in an undirected graph $\mathcal{H}$ is a path such that for every non-adjacent $p_i, p_j \in \mathcal{P}$ there is no edge $(p_i — p_j)$ in $\mathcal{H}$. An induced (non-chordal) cycle is an induced path whose endpoints are the same vertex. ∎

Definition 2.3: A triangulated or chordal graph is an undirected graph that has no induced cycles. Equivalently, it is an undirected graph in which every cycle of length greater than three contains a chord. ∎

1. The properties defining a tree-decomposition are equivalent to the corresponding *family preserving* and *running intersection* properties of clique trees introduced by Lauritzen and Spiegelhalter (1988) at around the same time.

It can be easily shown (Robertson and Seymour, 1987) that the treewidth of a given triangulated graph is the size of the maximal clique of the graph minus one. The treewidth of an undirected graph $\mathcal{H}$ is then equivalently the minimum treewidth over all possible triangulations of $\mathcal{H}$.

For the underlying directed acyclic graph of a Bayesian network, the treewidth can be characterized via a triangulation of the moralized graph.

Definition 2.4: A moralized graph $\mathcal{M}$ of a directed acyclic graph $\mathcal{G}$ is an undirected graph that includes an edge $(i—j)$ for every edge $(i \rightarrow j)$ in $\mathcal{G}$ and an edge $(p—q)$ for every pair of edges $(p \rightarrow i), (q \rightarrow i)$ in $\mathcal{G}$. ∎

The treewidth of a Bayesian network graph $\mathcal{G}$ is defined as the treewidth of its moralized graph $\mathcal{M}$, and corresponds to the complexity of inference in the model. It follows that the maximal clique of *any* moralized triangulation of $\mathcal{G}$ is an upper bound on the treewidth of the model, and thus its inference complexity.[2]

3. Learning Bounded Treewidth Bayesian Networks: Overview

Our goal is to develop an efficient algorithm for learning Bayesian networks with an arbitrary treewidth bound. As learning the optimal such network is NP-hard (Dagum and Luby, 1993), it is important to note the properties that we would like our algorithm to have. First, we would like our algorithm to be *provably* polynomial in the number of variables *and* in the desired treewidth. Thus, we cannot rely on methods such as that of Bodlaender (1996) to verify the boundedness of our network as they are super-exponential in the treewidth and are practical only for small treewidths. Second, we want to learn networks that are non-trivial. That is, we want to ensure that we do not quickly get stuck in local maxima due to the heuristic employed for bounding the treewidth of our model. Third, similar to the method of Chow and Liu (1968), we want to employ global structure operators that are optimal in some sense. In this section we present a brief high-level overview of our algorithm. In the next sections we provide detailed description of the different components along with proof of correctness and running time guarantees.

At the heart of our method is the idea of using a dynamically maintained moralized triangulated graph to upper bound the treewidth of the current Bayesian network. When an edge is added to the Bayesian network we update this (moralized) triangulated graph in a particular manner that is not only guaranteed to produce a valid triangulation, but that is also treewidth-friendly. That is, our update is guaranteed to increase the size of the maximal clique of the triangulated graph, and hence the treewidth bound, by at most one. As we will see, the correctness of our treewidth-friendly edge update as well as the fact that we can carry it out efficiently will both directly rely on the dynamic nature of our method. We discuss our edge update procedure in detail in Section 4.

An important property of our edge update is that we can characterize the parts of the network that are "contaminated" by the update by using the notion of blocks (bi-connected components) in the triangulated graph. This allows us to define sets of edges that are *jointly* treewidth-friendly. That is, these edge sets are guaranteed to increase the treewidth of the triangulated graph by at most one when all edges in the set are added to the Bayesian network structure. We discuss multiple edge updates in Section 5.

2. It also follows that the size of a family (a node and its parents) provides a lower bound on the treewidth, although we will not make use of this property in our work.

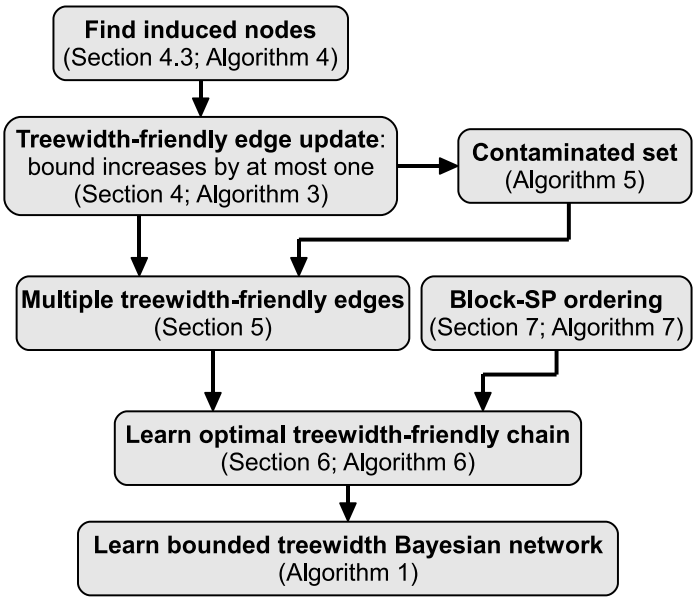


Figure 1: The building blocks of our method for learning Bayesian networks of bounded treewidth and how they depend on each other.

Building on the characterization of treewidth-friendly sets, we propose a dynamic programming approach for efficiently learning the optimal treewidth-friendly chain with respect to a node ordering. We present this procedure in Section 6. To encourage chains that are rich in structure (have many edges), in Section 7 we propose a block shortest-path node ordering that is motivated by the properties of our triangulation procedure.

Finally, we learn Bayesian networks with bounded treewidth by starting with a Chow-Liu tree (Chow and Liu, 1968) and iteratively applying a global structure modification operator where the current structure is augmented with a treewidth-friendly chain that is optimal with respect to the ordering chosen. Appealingly, as each global modification can increase our estimate of the treewidth by at most one, if our bound on the treewidth is K, at least K such chains will be added before we even face the problem of local maxima. In practice, as some chains do not increase the treewidth, many more such chains are added for a given maximum treewidth bound. Figure 1 illustrates the relationship between the different components of our approach.

Algorithm (1) shows pseudo-code of our method. Briefly, Line 4 initializes our model with a Chow and Liu (1968) tree; Line 8 produces a node ordering given the model at hand; Line 9 finds the optimal chain with respect to that ordering; and Line 10 augments the current model with the new edges. We then use our treewidth-friendly edge update procedure to perform the moralization and triangulation on $\mathcal{M}^+$ for each edge added to the Bayesian network $\mathcal{G}$ (Line 12). Once the maximal clique size reaches the treewidth bound K, we continue to add edges greedily until no more edges can be added without increasing the treewidth (Line 16).

Theorem 3.1: Given a treewidth bound K, Algorithm (1) runs in time polynomial in the number of variables and K.

Algorithm 1: Learning A Bayesian Network with Bounded Treewidth

Input : $\mathcal{D}$ // training set
K // maximum treewidth
Output: $\mathcal{G}$ // a graph structure with treewidth at most K

$\mathcal{G} \leftarrow$ maximum scoring spanning tree
$\mathcal{M}^+ \leftarrow$ undirected skeleton of $\mathcal{G}$
$k \leftarrow 1$
while $k < K$ *and positive scoring edges exist* **do**
  $O \leftarrow$ node ordering given $\mathcal{G}$ and $\mathcal{M}^+$ // Algorithm (7)
  $\mathcal{C} \leftarrow$ maximum scoring chain with respect to O // Algorithm (6)
  $\mathcal{G} \leftarrow \mathcal{G} \cup \mathcal{C}$
  foreach $(i \rightarrow j) \in \mathcal{C}$ **do**
    $\mathcal{M}^+ \leftarrow$ EdgeUpdate$(\mathcal{M}^+, (i \rightarrow j))$ // Algorithm (3)
  end foreach
  $k \leftarrow$ maximal clique size of $\mathcal{M}^+$
end
Greedily add edges to $\mathcal{G}$ that do not increase treewidth beyond K
return $\mathcal{G}$

We will prove this result gradually using the developments of the next sections. Note that we will show that our method is guaranteed to be polynomial both in the size of the graph *and* the treewidth bound. Thus, like the greedy thin junction tree approach of Bach and Jordan (2002), it can be used to learn a Bayesian networks given an arbitrary treewidth bound. It is also important to note that, as in the case of the thin junction tree method, the above result is only useful if the actual Bayesian network learned is expressive enough to be useful for generalization. As we will demonstrate in Section 8, by making use of global treewidth-friendly updates, our method indeed improves on the greedy approach and learns models that are rich and useful in practice.

4. Treewidth-Friendly Edge Update

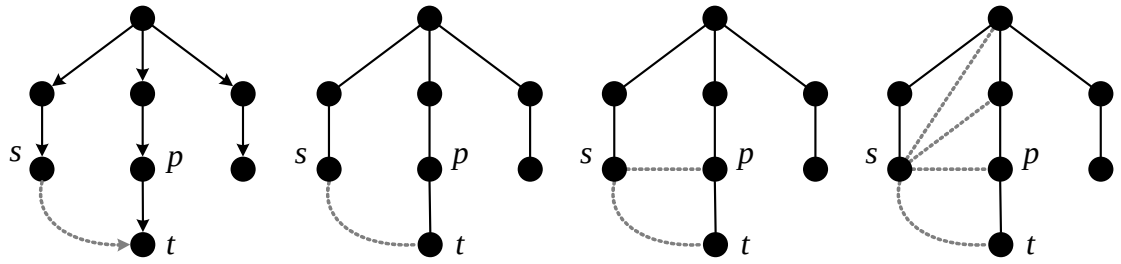
In this section we consider the basic building block of our method: the manner in which we update the triangulated graph when a single edge is added to the Bayesian network structure. Throughout this section we will build on the dynamic nature of our method and make use of the valid moralized triangulation graph that was constructed before adding an edge $(s \rightarrow t)$ to the Bayesian network structure. We will start by augmenting it with $(s$—$t)$ and any edges required for moralization. We will then triangulate the graph in a treewidth-friendly way, increasing the size of the maximal clique in the triangulated graph by at most one. For clarity of exposition, we start with a simple variant of our triangulation procedure in Section 4.1 and refine it in Section 4.2.

4.1 Single-source Triangulation

To gain intuition into how the dynamic nature of our update is useful, we use the notion of induced paths or paths with no shortcuts (see Section 2), and make explicit the following obvious fact.

Algorithm 2: SingleSourceEdgeUpdate: Update of $\mathcal{M}^+$ when adding $(s \rightarrow t)$ to $\mathcal{G}$

1 **Input** : $\mathcal{M}^+$ // triangulated moralized graph of $\mathcal{G}$
2 $(s \rightarrow t)$ // edge to be added to $\mathcal{G}$
3 **Output**: ${\mathcal{M}^+}_{(s \rightarrow t)}$ // a triangulated moralized graph of $\mathcal{G} \cup (s \rightarrow t)$
4 ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow \mathcal{M}^+ \cup (s—t)$
5 **foreach** $p \in \mathbf{Pa}_t$ **do**
6 ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s—p)$ // moralization
7 **end foreach**
8 **foreach** *node v on an induced path between s and* $t \cup \mathbf{Pa}_t$ *in* $\mathcal{M}^+$ **do**
9 ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s—v)$
10 **end foreach**
11 **return** ${\mathcal{M}^+}_{(s \rightarrow t)}$



(a) Bayesian network $\mathcal{G} \cup (s \rightarrow t)$

(b) Addition of $(s—t)$ to $\mathcal{M}^+$ (line 4)

(c) Addition of moralizing edges (lines 5-7)

(d) Addition of the triangulating edges (lines 8-10)

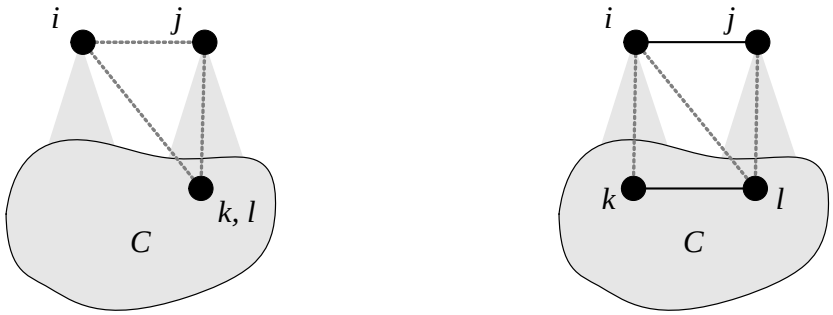
Figure 2: Example showing the application of the single-source triangulation procedure of Algorithm (2) to a simple graph. The treewidth of the original graph is one, while the graph augmented with $(s \rightarrow t)$ has a treewidth of two (maximal clique of size three).

Observation 4.1: Let $\mathcal{G}$ be a Bayesian network structure and let $\mathcal{M}^+$ be a moralized triangulation of $\mathcal{G}$. Let $\mathcal{M}_{(s \rightarrow t)}$ be $\mathcal{M}^+$ augmented with the edge $(s—t)$ and with the edges $(s—p)$ for every parent p of t in $\mathcal{G}$. Then, every non-chordal cycle in $\mathcal{M}_{(s \rightarrow t)}$ involves s and either t or a parent of t and an induced path between the two vertices.

Stated simply, if the graph was triangulated before the addition of $(s \rightarrow t)$ to the Bayesian network, then we only need to triangulate cycles created by the addition of the new edge or those forced by moralization. This observation immediately suggests the straight-forward *single-source triangulation* outlined in Algorithm (2): add an edge $(s—v)$ for every node v on an induced path between s and t or s and a parent p of t before the edge update. Figure 2 shows an application of the procedure to a simple graph. Clearly, this naive method results in a valid moralized triangulation of $\mathcal{G} \cup (s \rightarrow t)$. Surprisingly, we can also show that it is treewidth-friendly.

Theorem 4.2: The treewidth of the output graph ${\mathcal{M}^+}_{(s \rightarrow t)}$ of Algorithm (2) is greater than the treewidth of the input graph $\mathcal{M}^+$ by at most one.

Proof: Let C be the nodes in any maximal clique $\mathcal{M}^+$. We consider the minimal set of edges required to increase the size of C by more than one and show that this set cannot be a subset of the edges added by our single-source triangulation. In order for the clique to grow by more than one node, at least two nodes i and j not originally in C must become connected to all nodes in C. Since there exists at least one node $k \in C$ that is not adjacent to i and similarly there exists at least one node $l \in C$ not adjacent to j, both edges $(i—k)$ and $(j—l)$ are needed to form the larger clique. There are two possibilities illustrated below (the dotted edges are needed to increase the treewidth by two and all other edges between i, j and the current maximal clique are assumed to exist):

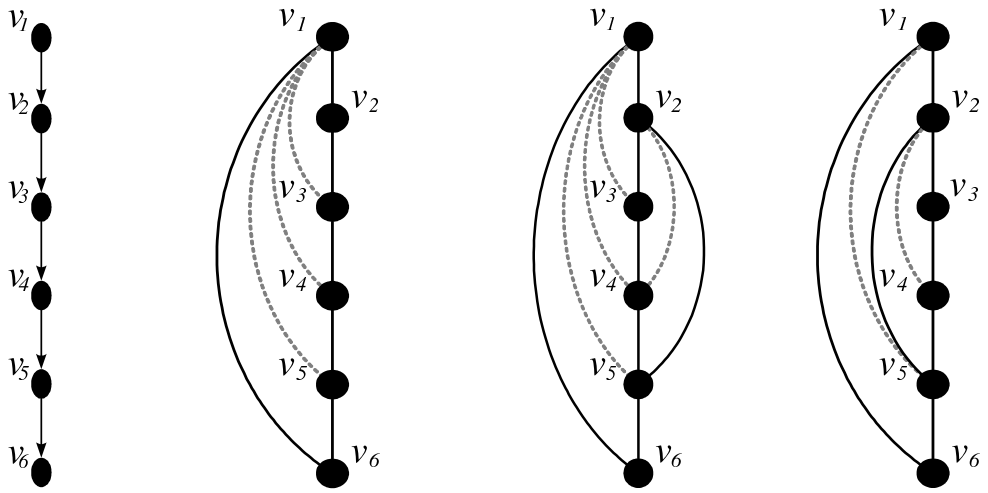


(a) $(i—j)$ does not exist (b) $(i—j)$ exists

- $(i—j)$ **does not exist (a)**. In this case k and l can be the same node but the missing edge $(i—j)$ is also required to form the larger clique.
- $(i—j)$ **exists (b)**. In this case k and l cannot be the same node or the original clique was not maximal since $C \cup i \cup j \setminus k$ would have formed a larger clique. Furthermore one of k or l must not be connected to both i and j otherwise $i—j—k—l—i$ forms a non-chordal cycle of length four contradicting our assumption that the original graph was triangulated. Thus, in this case either $(i—l)$ or $(j—k)$ are also required to form the larger clique.

In both scenarios, at least two nodes have two incident edges and the three edges needed cannot all be incident to a single vertex. Now consider the triangulation procedure. Since, by construction, all edges added in Algorithm (2) emanate from s, the above condition (requiring two nodes to have two incident edges and the three edges not all incident to a single vertex) is not met and the size of the maximal clique in the new graph cannot be larger than the size of the maximal clique in $\mathcal{M}^+$ by more than one. It follows that the treewidth of the moralized triangulated graph cannot increase by more than one. ∎

One problem with the proposed single-source triangulation, despite it being treewidth-friendly, is the fact that so many vertices are connected to the source node making the triangulations shallow (the length of the shortest path between any two nodes is small). While this is not a problem when considering a single edge update, it can have an undesirable effect on future edges and increases the chances of the formation of large cliques. As an example, Figure 3 shows a simple case where two successive single-source edge updates increase the treewidth by two while an alternative approach increases the treewidth by only one. In the next section, we present a refinement of the single-source triangulation that is motivated by this example.



(a) Chain network (b) Triangulation after $(v_1 \rightarrow v_6)$ is added (c) Triangulation after $(v_2 \rightarrow v_5)$ is added (d) Alternative triangulation

Figure 3: Example demonstrating that the single-source edge update of Algorithm (2) can be problematic for later edge additions. (a) shows a simple six nodes chain Bayesian network; (b) a single-source triangulation when $(v_1 \rightarrow v_6)$ is added to the network with a treewidth of two; (c) a single-source triangulation when in addition $(v_2 \rightarrow v_5)$ is added to the model with a treewidth of three; (d) an alternative triangulation to (b). This triangulation already includes the edge $(v_2$—$v_5)$ and the moralizing edge $(v_2$—$v_4)$ and thus is also a valid moralized triangulation after $(v_2 \rightarrow v_5)$ has been added, but has a treewidth of only two.

4.2 Alternating Cut-vertex Triangulation

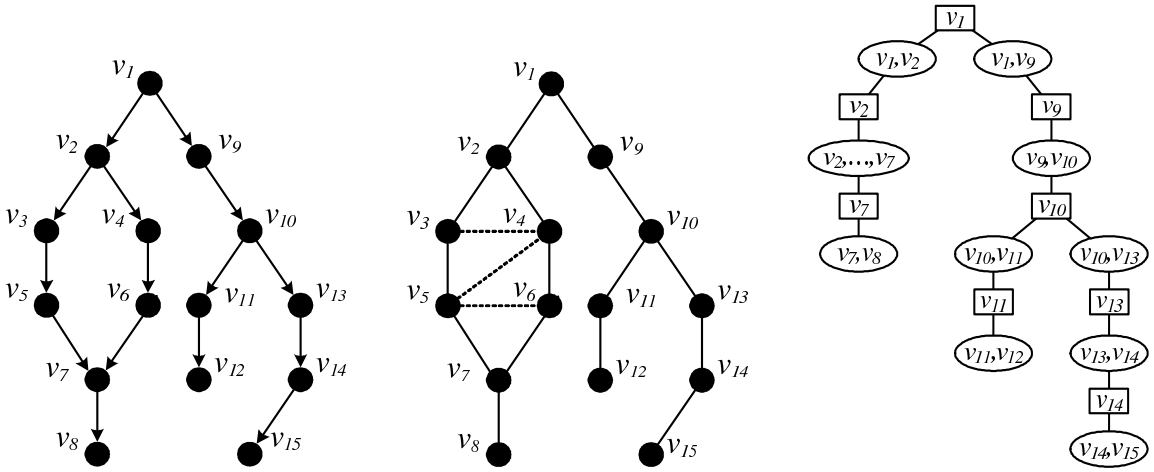
To refine the single-source triangulation discussed above with the goal of addressing the problem exemplified in Figure 3 we make use of the concepts of cut-vertices, blocks, and block trees (see, for example, Diestel, 2005).

Definition 4.3: A block, or biconnected component, of an undirected graph is a set of connected nodes that cannot be disconnected by the removal of a single vertex. By convention, if the edge $(u$—$v)$ is in the graph then u and v are in the same block. Vertices that separate (are in the intersection of) blocks are called cut-vertices. ∎

It follows directly from the definition that between every two nodes in a block (of size greater than two), there are at least two distinct paths, that is, a cycle. There are also no simple cycles involving nodes in different blocks.

Definition 4.4: A block tree $\mathcal{B}$ of an undirected graph $\mathcal{H}$ is a graph with nodes that correspond both to cut-vertices and to blocks of $\mathcal{H}$. The edges in the block tree connect any block node B_i with a cut-vertex node v_j if and only if $v_j \in B_i$ in $\mathcal{H}$. ∎

It can be easily shown that the above connectivity condition indeed forces a tree structure and that this tree is unique (see Figure 4 for an example). In addition, any path in $\mathcal{H}$ between two nodes



(a) Bayesian network, $\mathcal{G}$ (b) A possible triangulated graph, $\mathcal{M}^+$ (c) Unique block tree, $\mathcal{B}$

Figure 4: Example of a Bayesian network with a corresponding moralized triangulated graph and the unique block tree. Boxes in the block tree denote cut-vertices, ellipses denote blocks.

in different blocks passes through all the cut-vertices along the path between the blocks in $\mathcal{B}$. An important consequence that directly follows from the result of Dirac (1961) is that an undirected graph whose blocks are triangulated is overall triangulated.

We can now describe our improved treewidth-friendly triangulation outlined in Algorithm (3) and illustrated via an example in Figure 5. First, the triangulated graph is augmented by the edge $(s—t)$ and any edges needed for moralization (Figure 5(b) and (c)). Second, if s and t are not in the same block, a block level triangulation is carried out by starting from s and zig-zagging across the cut-vertices along the unique path between the blocks containing s and t and its parents in the block tree (Figure 5(d)). Next, within each block along the path (not containing s or t), a chord is added between the "entry" and "exit" cut-vertices along the *block path*, thereby short-circuiting any other *node path* through the block. In addition, within each such block we perform a single-source triangulation with respect to s' by adding an edge $(s'—v)$ between the first cut-vertex s' and any node v on an induced path between s' and the second cut-vertex t'. The block containing s is treated the same as other blocks on the path with the exception that the short-circuiting edge is added between s and the first cut-vertex along the path from s to t. For the block containing t and its parents, instead of adding a chord between the entry cut-vertex and t, we add chords directly from s to any node v (within the block) that is on an *induced path* between s and t (or parents of t) (Figure 5(e)). This is required to prevent moralization and triangulation edges from interacting in a way that will increase the treewidth by more than one (see Figure 5(f) for an example). If s and t happen to be in the same block, then we only triangulate the induced paths between s and t, that is, the last step outlined above. Finally, in the special case that s and t are in *disconnected* components of $\mathcal{G}$, the only edges added are those required for moralization.

We now show that this revised edge update is a valid triangulation procedure and that it is also treewidth-friendly. To do so we start with the following observations that are a direct consequence of the definition of a block and block tree.

Algorithm 3: EdgeUpdate: Update of $\mathcal{M}^+$ when adding $(s \rightarrow t)$ to $\mathcal{G}$

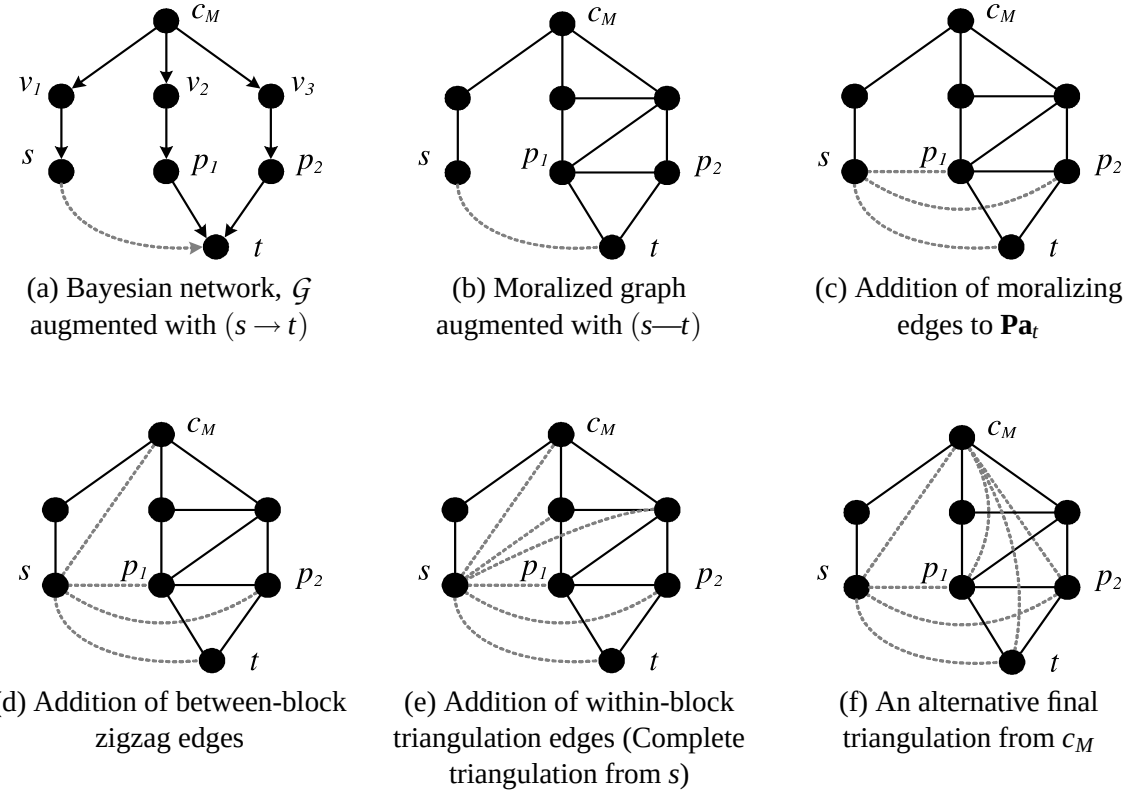
Input : $\mathcal{M}^+$ // triangulated moralized graph of $\mathcal{G}$
O // node ordering
$(s \rightarrow t)$ // edge to be added to $\mathcal{G}$
Output: ${\mathcal{M}^+}_{(s \rightarrow t)}$ // a triangulated moralized graph of $\mathcal{G} \cup (s \rightarrow t)$

$\mathcal{B} \leftarrow$ block tree of $\mathcal{M}^+$
${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow \mathcal{M}^+ \cup (s — t)$
foreach $p \in \mathbf{Pa}_t$ **do**
  ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s — p)$ // moralization
end foreach
// triangulate (cut-vertices) between blocks
$\mathcal{C} = \{c_1, \ldots, c_M\} \leftarrow$ sequence of cut-vertices on the path from s to $t \cup \mathbf{Pa}_t$ in block tree $\mathcal{B}$
Add $(s — c_M), (c_M — c_1), (c_1 — c_{M-1}), (c_{M-1} — c_2), \ldots$ to ${\mathcal{M}^+}_{(s \rightarrow t)}$
// triangulate nodes within blocks on path from s to $t \cup \mathbf{Pa}_t$
$\mathcal{E} \leftarrow \{(s — c_1), (c_1 — c_2), \ldots, (c_{M-1} — c_M)\}$
foreach *edge* $(s' — t') \in \mathcal{E}$ **do**
  ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s' — t')$
  foreach *node v on an induced path between s' and t' in the* **original** *block containing both* **do**
    ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s' — v)$
  end foreach
end foreach
// triangulate s with nodes in block containing $t \cup \mathbf{Pa}_t$
foreach *node v on an induced path between s and $t \cup \mathbf{Pa}_t$ in the* **new** *block containing them* **do**
  ${\mathcal{M}^+}_{(s \rightarrow t)} \leftarrow {\mathcal{M}^+}_{(s \rightarrow t)} \cup (s — v)$
end foreach
return ${\mathcal{M}^+}_{(s \rightarrow t)}$

Observation 4.5: (Family Block). Let u be a node in a Bayesian network $\mathcal{G}$ and let $\mathbf{Pa}_u$ be the set of parents of u. Then the block tree for any moralized triangulated graph $\mathcal{M}^+$ of $\mathcal{G}$ has a unique block containing $\{u, \mathbf{Pa}_u\}$.

Observation 4.6: (Path Nodes). Let $\mathcal{B} = (\{B_i\} \cup \{c_j\}, \mathcal{T})$ be the block tree of $\mathcal{M}^+$ with blocks $\{B_i\}$ and cut-vertices $\{c_j\}$. Let s and t be nodes in blocks B_s and B_t, respectively. If t is a cut-vertex then let B_t be the (unique) block that also contains $\mathbf{Pa}_t$. If s is a cut-vertex, then choose B_s to be the block containing s closest to B_t in $\mathcal{T}$. Then a node v is on a path from s to t or from s to a parent of t if and only if it is in a block that is on the unique path from B_s to B_t.

Figure 4(c) shows an example of a block tree for a small Bayesian network. Here, for example, selecting s to be the node v_6 and t to be the node v_{10} in $\mathcal{G}$, it is clear that all paths between s and t include only the vertices that are in blocks along the unique block path between B_s and B_t. Furthermore, every path between s and t passes through all the cut-vertices on this block path, that



(a) Bayesian network, $\mathcal{G}$ augmented with $(s \to t)$

(b) Moralized graph augmented with $(s—t)$

(c) Addition of moralizing edges to $\mathbf{Pa}_t$

(d) Addition of between-block zigzag edges

(e) Addition of within-block triangulation edges (Complete triangulation from s)

(f) An alternative final triangulation from c_M

Figure 5: Example showing our triangulation procedure (b)-(e) for s and t in different blocks. (The blocks are $\{s, v_1\}$, $\{v_1, c_M\}$, and $\{c_M, v_2, v_3, p_1, p_2, t\}$ with corresponding cut-vertices v_1 and c_M). The original graph has a treewidth of two, while the graph augmented with $(s \to t)$ has treewidth three (maximal clique of size four). An alternative triangulation (f), connecting c_M to t, however, would result in a clique of size five $\{s, c_M, p_1, p_2, t\}$.

is, $\{v_2, v_1, v_9\}$. We can now use these properties to show that our edge update procedure produces a valid triangulation.

Lemma 4.7: *If $\mathcal{M}^+$ is a valid moralized triangulation of the graph $\mathcal{G}$ then Algorithm (3) produces a moralized triangulation $\mathcal{M}^+{}_{(s \to t)}$ of the graph $\mathcal{G}_{(s \to t)} \equiv \mathcal{G} \cup (s \to t)$.*

Proof: Since $\mathcal{M}^+$ was triangulated, every cycle of length greater than or equal to four in $\mathcal{G}_{(s \to t)}$ is the result of the edge $(s—t)$ or one of the moralizing edges, together with an induced path (path with no shortcuts) between the endpoints of the edge. We consider three cases:

- s **and** t **are disconnected in** $\mathcal{M}^+$**:** There are no induced paths between s and t so the only edges required are those for moralization. These edges do not produce any induced cycles.

- s **and** t **are in the same block:** The edge $(s—t)$ does not create a new block and all simple cycles that involve both s and t must be within the block. Thus, by construction, the edges added in Line 16 triangulate all newly introduced induced cycles. If the parents of t are in the same block as s and t, the same reasoning holds for all induced paths between a parent p of

t and s. Otherwise, t is a cut-vertex between the block that contains its parents and the block that contains s. It follows that all paths (including induced ones) from a parent of t to s pass through t and the edges added for the s,t-block triangulate all newly created induced cycles that result from the moralizing edges.

- **s and t are not in the same block:** As noted in Observation 4.6, all paths in $\mathcal{M}^+$ from s to t or a parent of t pass through the unique cut-vertex path from the block containing s to the block containing t and its parents. The edges added in Line 14 short-circuit the in-going s' and out-going t' of each block creating a path containing only cut-vertices between s and t. Line 11 triangulates this path by forming cycles of length three containing s', t' and some other cut-vertex. The only induced cycles remaining are contained within blocks and contain the newly added edge $(s'—t')$ or involve the edge between s and the last cut-vertex $(s—c_M)$ and one of the edges between s and t or a parent of t. It follows that within-block triangulation with respect to s' and t' will shortcut the former induced cycles, and the edges added from s in Line 20 will shortcut the later induced cycles.

To complete the proof, we need to show that any edge added from s (or s') to an induced node v does not create new induced cycles. Any such induced cycle would have to include an induced path from the endpoints of the edge added and thus would have been a sub-path of some induced cycle that includes both s and v. This cycle would have already been triangulated by our procedure. ∎

Having shown that our update produces a valid triangulation, we now prove that our edge update is indeed treewidth-friendly and that it can increase the treewidth of the moralized triangulated graph by at most one.

Theorem 4.8: The treewidth of the output graph $\mathcal{M}^+{}_{(s\rightarrow t)}$ of Algorithm (3) is greater than the treewidth of the input graph $\mathcal{M}^+$ by at most one.

Proof: As shown in the proof of Theorem 4.2, the single-source triangulation within a block is guaranteed not to increase the maximal clique size by more than one. In addition, from the properties of blocks it follows directly that the inner block triangulation does not add edges that are incident to nodes outside of the block. It follows that all the inner block single-source triangulations independently effect disjoint cliques. Thus, the only way that the treewidth of the graph can further increase is via the zig-zag edges. Now consider two cliques in different blocks. Since our block level zig-zag triangulation only touches two cut-vertices in each block, it cannot join two cliques of size greater than two into a single larger one. In the simple case of two blocks with two nodes (a single edge) and that intersect at a single cut-vertex, a zig-zag edge can indeed increase the treewidth by one. In this case, however, there is no within-block triangulation and so the overall treewidth cannot increase by more than one. ∎

4.3 Finding Induced Nodes

We finish the description of our edge update (Algorithm (3)) by showing that it can be carried out efficiently. That is, we have to be able to efficiently find the vertices on *all* induced paths between two nodes in a graph. In general, this task is computationally difficult as there are potentially exponentially many such paths between any two nodes. To cope with this problem, we again make use of the dynamic nature of our method.

The idea is simple. As implied by Observation 4.1, any induced path between s' and t' in a triangulated graph will be part of an induced cycle if $(s'—t')$ is added to the graph. Furthermore,

Algorithm 4: InducedNodes: compute set of nodes on induced path between s' and t' in $\mathcal{M}^+$

```
1  Input  : M+    // moralized triangulated graph
2           s',t' // two nodes in M+
3  Output: I      // set of nodes on induced paths between s' and t'
4  H ← block (subgraph) of M+ ∪ (s'—t') containing s' and t'
5  I ← ∅
6  while edges being added do
       // maximum cardinality search
7      X ← all nodes in H except s'
8      Y ← {s'}
9      while X ≠ ∅ do
10         Find v ∈ X with maximum number of neighbors in Y
11         X ← X \ {v} and Y ← Y ∪ {v}   // remove from X, add to Y
12         if there exists u,w ∈ Y such that (u—w) ∉ H then
13             I ← I ∪ {u,v,w}
14             Add edges (s'—u), (s'—v) and (s'—w) to H
15             Restart maximum cardinality search
16         end
17     end
18 end
19 return I
```

after adding $(s'—t')$ to the graph, *every* cycle detected will involve an induced path between the two nodes. Using this observation, we can make use of the ability of the maximum cardinality search algorithm (Tarjan and Yannakakis, 1984) to iteratively detect non-chordal cycles.

The method is outlined in Algorithm (4). At each iteration we attempt to complete a maximum cardinality search starting from s' (Line 7 to Line 17). If the search fails, we add the node at which it failed, v, together with its non-adjacent neighboring nodes $\{u,w\}$ to the set of induced nodes and augment the graph with triangulating edges from s' to each of $\{u,v,w\}$. If the search completes then we have successfully triangulated the graph and hence found all induced nodes. Note that using the properties of blocks and cut-vertices, we only need to consider the subgraph that is the block created after the addition of $(s'—t')$ to the graph.

Lemma 4.9: (Induced Nodes). *Let $\mathcal{M}^+$ be a triangulated graph and let s' and t' be any two nodes in $\mathcal{M}^+$. Then Algorithm (4) efficiently returns all nodes on any induced path between s' and t' in $\mathcal{M}^+$, unless those nodes are connected directly to s'.*

Proof: During a maximum cardinality search, if the next node chosen v has two neighbors u and w that are not connected then the triplet $u—v—w$ is part of an induced cycle. As the graph was triangulated before adding the edge $(s'—t')$, all such cycles must contain s' and adding $(s'—v)$ obviously shortcuts such a cycle. This is also true for v and v' that are on the same induced cycle. It remains to show that the edges added do not create new induced cycle. Such an induced cycle would have to include the edge $(s'—v)$ as well as an induced path between s' and v. However, such

a path must have been part of another cycle where v was an induced node and hence would have been triangulated. ∎

Thus Algorithm (4) returns exactly the set of nodes on induced paths from s' to t' that s' needs to connect to in order to triangulate the graph $\mathcal{M}^+ \cup (s—t)$. The efficiency of our edge update procedure of Algorithm (3) follows immediately as all other operations are simple.

5. Multiple Edge Updates

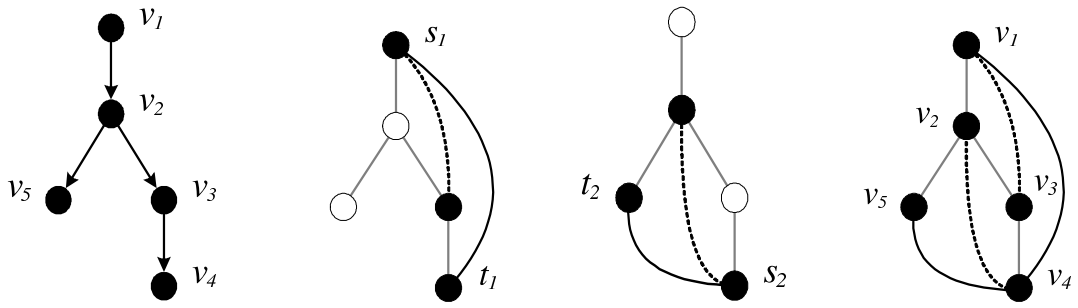
In this section we define the notion of a *contaminated set*, or the subset of nodes that are incident to edges added to $\mathcal{M}^+$ in Algorithm (3), and characterize sets of edges that are jointly guaranteed not to increase the treewidth of the triangulated graph by more than one. We begin by formally defining the terms *contaminate* and *contaminated set*.

Definition 5.1: We say that a node v is contaminated by the addition of the edge $(s \rightarrow t)$ to $\mathcal{G}$ if it is incident to an edge added to the moralized triangulated graph $\mathcal{M}^+$ by a call to Algorithm (3). The contaminated set for an edge $(s \rightarrow t)$ is the set of all nodes v that would be contaminated (with respect to $\mathcal{M}^+$) by adding $(s \rightarrow t)$ to $\mathcal{G}$, including s, t, and the parents of t. ∎

Figure 6 shows some examples of contaminated sets for different edge updates. Note that our definition of contaminated set only includes nodes that are incident to *new* edges added to $\mathcal{M}^+$ and, for example, excludes nodes that were already connected to s before $(s \rightarrow t)$ is added, such as the two nodes adjacent to s in Figure 6(b).

Using the separation properties of cut-vertices, one might be tempted to claim that if the contaminated sets of two edges overlap at most by a single cut-vertex then the two edges jointly increase the treewidth by at most one. This however, is not true in general as the following example shows.

Example 5.2: Consider the Bayesian network shown below in (a) and its triangulation (b) after $(v_1 \rightarrow v_4)$ is added, increasing the treewidth from one to two. (c) is the same for the case when $(v_4 \rightarrow v_5)$ is added to the network. Despite the fact that the contaminated sets (solid nodes) of two edge additions overlap only by the cut-vertex v_4, (d) shows that jointly adding the two edges to the graph results in a triangulated graph with a treewidth of three. ∎



(a) (b) (c) (d)

The problem in the above example lies in the overlap of *block paths* between the endpoints of the two edges, a property that we have to take into account while characterizing sets of treewidth-friendly edges.

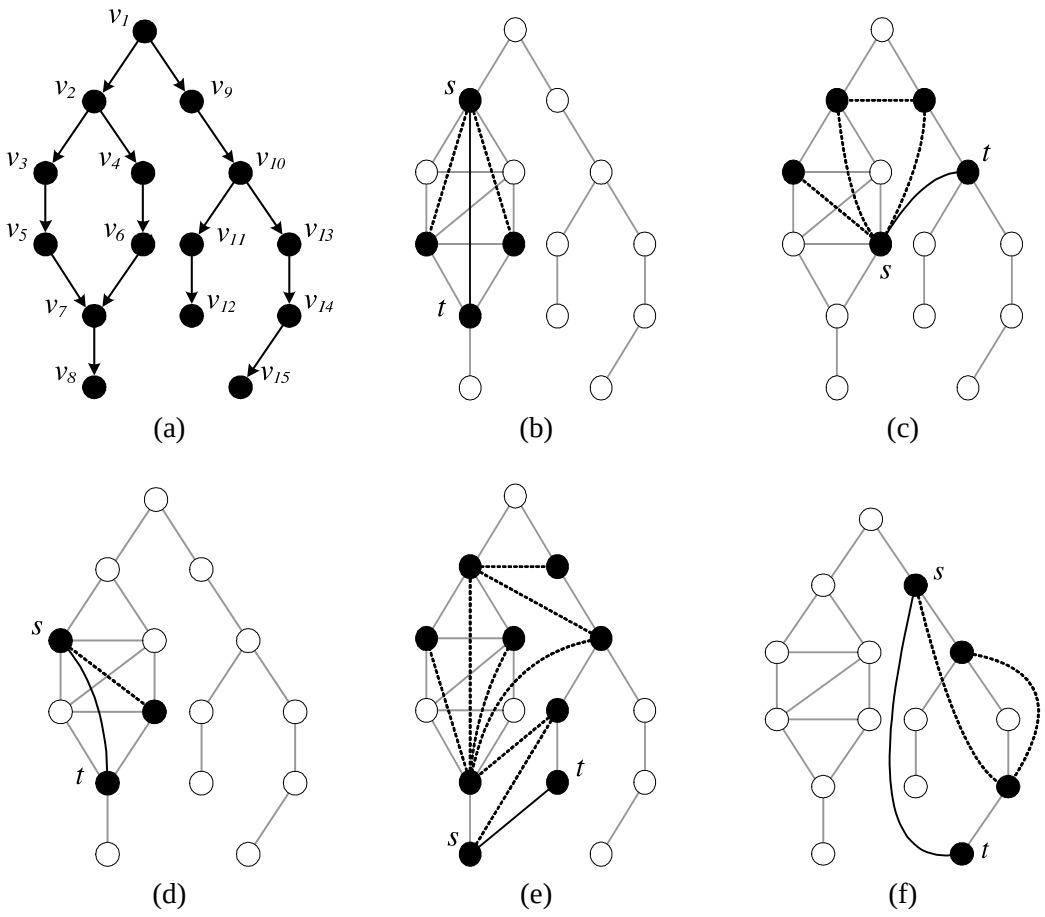


Figure 6: Some examples of contaminated sets (solid nodes) that are incident to edges added (dashed) by Algorithm (3) for different candidate edge additions $(s \rightarrow t)$ to the Bayesian network shown in (a). In (b), (c), (d), and (e) the treewidth is increased by one; In (f) the treewidth does not change.

Theorem 5.3: (Treewidth-friendly pair). Let $\mathcal{G}$ be a Bayesian network graph structure and $\mathcal{M}^+$ be its corresponding moralized triangulation. Let $(s \rightarrow t)$ and $(u \rightarrow v)$ be two distinct edges that are topologically consistent with $\mathcal{G}$. Then the addition of the edges to $\mathcal{G}$ does not increase the treewidth of $\mathcal{M}^+$ by more than one if *one* of the following conditions holds:

- the contaminated sets of $(s \rightarrow t)$ and $(u \rightarrow v)$ are disjoint.
- the endpoints of each of the two edges are not in the same block *and* the block paths between the endpoints of the two edges do not overlap *and* the contaminated sets of the two edge overlap at a single cut-vertex.

Proof: As in the proof of Algorithm (3) a maximal clique can grow by two nodes only if three undirected edges are added so that at least two nodes are incident to two of them. Obviously, this

Algorithm 5: ContaminatedSet: compute contaminated set for $(s \rightarrow t)$

Input : $\mathcal{G}$ // Bayesian network
$\mathcal{M}^+$ // moralized triangulated graph
$(s \rightarrow t)$ // candidate edge
Output: $\mathcal{C}_{s,t}$ // contaminated set for $(s \rightarrow t)$

$\mathcal{C}_{s,t} \leftarrow \{s,t\} \cup \{p \in \mathbf{Pa}_t \mid (s\text{—}p) \notin \mathcal{M}^+\}$
foreach *edge* $(s'\text{—}t') \in \mathcal{E}$ *in procedure Algorithm (3) with* $(s'\text{—}t') \notin \mathcal{M}^+$ **do**
  $I = \text{InducedNodes}(\mathcal{M}^+, \{s',t'\})$
  $\mathcal{C}_{s,t} \leftarrow \mathcal{C}_{s,t} \cup \{v \in I \mid (s'\text{—}v) \notin \mathcal{M}^+\}$
end foreach
$\mathcal{H} \leftarrow \{s\}$ and block containing $t \cup \mathbf{Pa}_t$
$\mathcal{H} \leftarrow \mathcal{H} \cup \{(s\text{—}p) \mid p \in \mathbf{Pa}_t\} \cup (s\text{—}c)$ where c is the cut-vertex closest to s in the block containing t
$I = \text{InducedNodes}(\mathcal{H}, \{s,t\})$
$\mathcal{C}_{s,t} \leftarrow \mathcal{C}_{s,t} \cup \{v \in I \mid (s'\text{—}v) \notin \mathcal{M}^+\}$
return $\mathcal{C}_{s,t}$

can only happen if the contamination sets of the two edge updates are not completely disjoint. Now, consider the case when the two sets overlap by a single cut-vertex. By construction all triangulating edges added are along the block path between the endpoints of each edge. Since the block paths of the two edge updates do not overlap there can not be an edge between a node in the contaminated set of $(s \rightarrow t)$ and the contaminated set of $(u \rightarrow v)$ (except for the single cut-vertex). But then no node from either contaminated set can become part of a clique involving nodes from the other contaminated set. Thus there are no two nodes that can be added to the same clique. It follows that the maximal clique size of $\mathcal{M}^+$, and hence the treewidth bound, cannot grow by more than one. ∎

The following result is an immediate consequence.

Corollary 5.4: (Treewidth-friendly set). *Let $\mathcal{G}$ be a Bayesian network graph structure and $\mathcal{M}^+$ be its corresponding moralized triangulation. If $\{(s_i \rightarrow t_i)\}$ is a set of edges so that every pair of edges satisfies the condition of Theorem 5.3 then adding all edges to $\mathcal{G}$ can increase the treewidth bound by at most one.*

The above result characterizes treewidth-friendly sets. In the search for such sets that are useful for generalization (see Section 6), we will need be able to efficiently compute the contaminated set of candidate edges. At the block level, adding an edge between s and t in $\mathcal{G}$ can only contaminate blocks between the block containing s and that containing t and its parents in the block tree for $\mathcal{M}^+$ (Observation 4.6). Furthermore, identifying the nodes that are incident to moralizing edges and edges that are part of the zigzag block level triangulation is easy. Finally, within a block, the contaminated set is easily computed using Algorithm (4) for finding the induced nodes between two vertices. Algorithm (5) outlines this procedure. Its correctness follows directly from the correctness of Algorithm (4) and the fact that it mirrors the edge update procedure of Algorithm (3).

6. Learning Optimal Treewidth-Friendly Chains

We now want to build on the results of the previous sections to facilitate the addition of global moves that are both optimal in some sense and are guaranteed to increase the treewidth by at most one. Specifically, we consider adding optimal chains that are consistent with some topological node ordering. On the surface, one might question the need for a specific node ordering altogether if chain global operators are to be used—given the result of Chow and Liu (1968), one might expect that learning the optimal chain with respect to *any* ordering can be carried out efficiently. However, Meek (2001) showed that learning such an optimal chain over a set of random variables is computationally difficult. Furthermore, conditioning on the current model, the problem of identifying the optimal chain is equivalent to learning the (unconditioned) optimal chain.[3] Thus, during any iteration of our algorithm, we cannot expect to find the overall optimal chain.

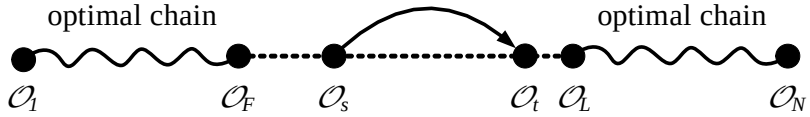
Instead, we commit to a single node ordering that is topologically consistent and learn the optimal chain *with respect to that order*. In this section we will complete the development of our algorithm and show how we can efficiently learn chains that are optimal with respect to any such ordering. In Section 7 we will also suggest a useful node ordering motivated by the characteristics of contaminated sets. We start by formally defining the chains that we will learn.

Definition 6.1: A treewidth-friendly chain $\mathcal{C}$ with respect to a node ordering O is a chain with respect to O such that the contamination conditions of Theorem 5.3 hold for the set of edges in $\mathcal{C}$. ∎

Given a treewidth-friendly chain $\mathcal{C}$ to be added for Bayesian network $\mathcal{G}$, we can apply the edge update of Algorithm (3) successively to every edge in $\mathcal{C}$ to produce a valid moralized triangulation of $\mathcal{G} \cup \mathcal{C}$. The result of Theorem 5.4 ensures that the resulting moralized triangulation will have treewidth at most one greater than the original moralized triangulation $\mathcal{M}^+$.

To find the optimal treewidth-friendly chain in polynomial time, we use a straightforward dynamic programming approach: the best treewidth-friendly chain that contains $(O_s \rightarrow O_t)$ is the concatenation of

- the best treewidth-friendly chain from the first node in the order O_1 to O_F, the first ordered node contaminated by the edge $(O_s \rightarrow O_t)$
- the edge $(O_s \rightarrow O_t)$
- the best treewidth-friendly chain starting from O_L, the last node contaminated by the edge $(O_s \rightarrow O_t)$, to the last node in the order, O_N.



We note that when the end nodes are not separating cut-vertices, we maintain a gap so that the contamination sets are disjoint and the conditions of Theorem 5.3 are met.

3. Consider, for example, the star-network where a single node acts as parent to all other nodes (with no other edges), then learning the optimal chain amounts to learning a chain over the $n-1$ children.

Formally, we define $C[i,j]$ as the optimal chain whose contamination starts at or after O_i and ends at or before O_j. To find the optimal treewidth-friendly chain with respect to a node ordering O for a Bayesian network with N variables, our goal is then to compute $C[1,N]$. Using the shorthand notation F to denote the first node ordered in the contamination set of $(s \rightarrow t)$ and L to denote the last ordered node in the contamination set, we can readily compute $C[1,N]$ via the following recursive update principle

$$C[i,j] = \max \begin{cases} \max_{s,t:F=i,L=j}(s \rightarrow t) & \text{no split} \\ \max_{k=i+1:j-1} C[i,k] \cup C[k,j] & \text{split} \\ \emptyset & \text{leave a gap} \end{cases}$$

where the maximization is with respect to the score (e.g., BIC) of the structures considered. In simple words, the maximum chain in any sub-sequence $[i,j]$ in the node ordering is the maximum of three alternatives: all edges whose contamination boundaries are exactly i and j (no split); all two chain combinations that are in the sub-sequence $[i,j]$ and are joined at some node $i < k < j$ (split); a gap between i and j in the case that there is no edge whose contamination is contained in this range and that increases the score.

Algorithm (6) outlines a simple backward recursion that computes $C[1,N]$. At each node, the algorithm maintains a list of the best partial chains evaluated so far that contaminates nodes up to, but not preceding, that node in the ordering. That is, the list of best partial chains is indexed by where the contamination boundary of each chain starts in the ordering. By recursing backwards from the last node, the algorithm is able to update this list by evaluating all candidate edges *terminating* at the current node. It follows that, once the algorithm iterates past a node t we have the optimal chain *starting* from that node. Thus, at the end of the recursion we are left with the optimal non-contaminating chain starting from the first node in the ordering.

The recursion starts at Line 7. If for node O_t the best chain starting from the succeeding node O_{t+1} is better than the best chain starting from O_t, we replace the best chain from O_t with the one from O_{t+1} simply leaving a gap in the chain (Line 8). Then, for every edge terminating at O_t, we find the first ordered node O_F and the last ordered node O_L that would be contaminated by adding that edge. If the score for the edge plus the score for the best partial non-contaminating chain from O_F is better than the current best partial chain from O_L then we replace the best chain from O_L with the one just found (Line 19).

With the ability to learn optimal chains with respect to a node ordering, we have completed the description of all the components of our algorithm for learning bounded treewidth Bayesian network outlined in Algorithm (1). Its efficiency is a direct consequence of our ability to learn treewidth-friendly chains in time that is polynomial both in the number of variables and in the treewidth at each iteration. For completeness we now restate and prove Theorem 3.1.

Theorem 3.1: Given a treewidth bound K, Algorithm (1) runs in time polynomial in the number of variables and K.

Proof: The initial Chow-Liu tree and its corresponding undirected skeleton can be obtained in polynomial time using a standard max-spanning-tree algorithm. The maximum scoring chain can be computed in polynomial time (using Algorithm (6)) at each iteration. As we proved, the same is true of the triangulation procedure of Algorithm (3). All other steps are trivial. Since the algorithm adds at least one edge per iteration it cannot loop for more than $K \cdot N$ iterations before exceeding a treewidth of K (where N is the number of variables). ∎

Algorithm 6: LearnChain: learn optimal non-contaminating chain with respect to topological node ordering

Input : $\mathcal{O}$ // topological node ordering
Output: $\mathcal{C}$ // non-contaminating chain

// initialize dynamic programming data
for $i = 1$ **to** $|\mathcal{O}| + 1$ **do**
  bestChain$[i] \leftarrow \emptyset$ // best chain from i-th node
  bestScore$[i] \leftarrow 0$ // best score from i-th node
end

// backward recursion
for $t = |\mathcal{O}|$ **down to** 1 **do**
  if (bestScore$[t+1] >$ bestScore$[t]$) **then**
    bestChain$[t] \leftarrow$ bestChain$[t+1]$
    bestScore$[t] \leftarrow$ bestScore$[t+1]$
  end
  for $s = 1$ **to** $t-1$ **do** // evaluate edges
    $\mathcal{V} \leftarrow$ contaminated set for candidate edge $(O_s \rightarrow O_t)$
    $f \leftarrow$ first ordered node in $\mathcal{V}$ // must be $\leq s$
    $l \leftarrow$ last ordered node in $\mathcal{V}$ // must be $\geq t$
    if bestChain$[l]$.last *and* $(O_s \rightarrow O_t)$ *do not satisfy the conditions of Theorem 5.3* **then**
      $l \leftarrow l+1$ // leave a gap
    end
    if ($\Delta Score(O_s \rightarrow O_t) +$ bestScore$[l] >$ bestScore$[f]$) **then**
      bestChain$[f] \leftarrow \{(O_s \rightarrow O_t)\} \cup$ bestChain$[l]$
      bestScore$[f] \leftarrow \Delta Score(O_s \rightarrow O_t) +$ bestScore$[l]$
    end
  end
end

// return optimal non-contaminating chain
return bestChain$[1]$

7. Block-Shortest-Path Ordering

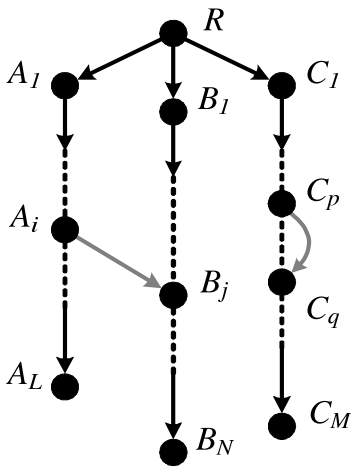
In the previous sections we presented an algorithm for learning bounded treewidth Bayesian networks given any topological ordering of the variables. In order to make the most of our method, we would like our ordering to facilitate rich structures that will have beneficial generalization properties. Toward that end, in this section we consider the practical matter of a concrete node ordering. We will present a block shortest-path (BSP) node ordering that is motivated by the specific properties of our triangulation method.[4]

4. We also considered several other strategies for ordering the variables. As none was better than the intuitive ordering described here, we only present results for our block-shortest-path ordering.

To make our node ordering concrete, since the contamination resulting from edges added within an existing block is limited to the block, we start by grouping together all nodes that are within a block (cut-vertices that appear in multiple blocks are included in the first block chosen). Our node ordering is then a topologically consistent ordering over the blocks combined with a topologically consistent ordering over the nodes within each block. We use topological consistency to facilitate as many edges as possible though this is not required by the theory (and, in particular, Theorem 5.3).

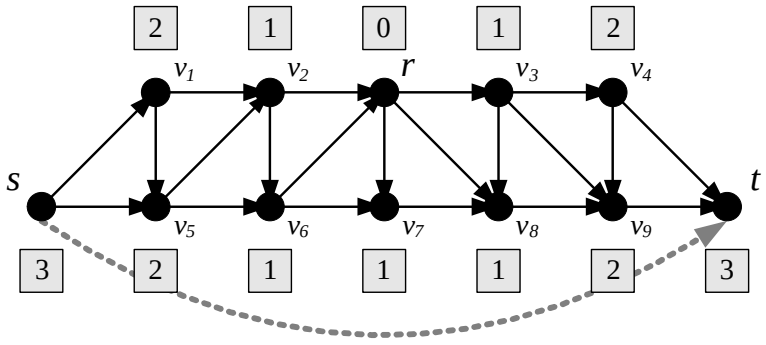
We now consider how to order interchangeable blocks by taking into account that our triangulation following an edge addition $(s \to t)$ only involves variables that are in blocks along the unique path between the block containing s and the block containing t and its parents. The following example motivates a natural choice for this ordering.

Example 7.1: Consider a Bayesian network with root node R and three branches: $R \to A_1 \to \ldots \to A_L$, $R \to B_1 \to \ldots \to B_N$, and $R \to C_1 \to \ldots \to C_M$. If we add an edge $A_i \to B_j$ to the network, then by the block contamination results, our triangulation procedure will touch (almost) every node on the path between A_i and B_j. This implies that we can not include additional edges of the type $B_k \to C_l$ in our chain since the block path from B_k to R overlaps with the block path from B_j to R. Note, however, that any edge $C_p \to C_{q>p}$ is still allowed to be added since its contaminated set does not overlap with that of $A_i \to B_j$. Now, consider the two obvious topological node orderings: $O^{\text{BFS}} = (R, A_1, B_1, C_1, A_2, \ldots)$ and $O^{\text{DFS}} = (R, A_1, \ldots, A_L, B_1, \ldots, B_N, C_1, \ldots)$. Only the DFS ordering, obtained by grouping the B_i's together, allows us to consider such edge combinations. ∎



Motivated by the above example to order interchangeable blocks, we use a block level depth-first ordering. The question now is whether a further characterization of the contaminated set can be provided in order to better order topologically interchangeable nodes within a block. To answer this question we consider the following example.

Example 7.2: Consider the Bayesian network shown below whose underlying undirected structure is a valid moralized triangulation and forms a single block. Numbers in the boxes indicate the (undirected) distance of each node from r, a property that we make use of below.



The single edge addition $(s \rightarrow t)$ will contaminate every node in the block (other than those already adjacent to it) since all nodes lie on induced paths between s and t. However other edge additions, such as $(v_3 \rightarrow t)$ have a much smaller contamination set: $\{v_3, t\}$. ∎

Based on the above example, one may think that no within-block ordering can improve the expected contamination of edges added, and that we may be forced to only add a single edge per block, making our method greedier than we would like. Fortunately, there is a straightforward way to characterize the within-block contamination set using the notion of shortest path length. Let $\mathcal{G}$ be a Bayesian network over variables $\mathcal{X}$. We denote by $d^M_{\min}(u,v)$ the minimum distance (shortest path) between nodes $u, v \in \mathcal{X}$ in $\mathcal{M}^+$. We note the following useful properties of $d^M_{\min}(\cdot,\cdot)$:

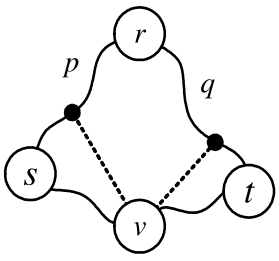
- $d^M_{\min}(u,v) \geq 0$ with equality if and only if $u = v$
- $d^M_{\min}(u,w) + d^M_{\min}(v,w) \geq d^M_{\min}(u,v)$ with equality if and only if w is on the (possibly non-unique) shortest path between u and v
- if u and v are disconnected in $\mathcal{M}^+$ then, by convention, $d^M_{\min}(u,v) = \infty$

Theorem 7.3: Let r, s and t be nodes in some block B (of size ≥ 3) in the triangulated graph $\mathcal{M}^+$ with $d^M_{\min}(r,s) \leq d^M_{\min}(r,t)$. Then for any v on an induced path between s and t we have

$$d^M_{\min}(r,v) \leq d^M_{\min}(r,t)$$

Proof: Since the nodes are all in the same block we know that there must be at least two paths between any two nodes. Let p and q be the shortest paths from nodes r to s and r to t, respectively (denoted $r \xrightarrow{p} s$ and $r \xrightarrow{q} t$). If p and q meet at some node other than r then they will share the path from that node to r (otherwise they cannot be shortest paths). Let such a shared node furthest from r be r'. Then $d^M_{\min}(r,t) = d^M_{\min}(r,r') + d^M_{\min}(r',t)$ and $d^M_{\min}(r,v) \leq d^M_{\min}(r,r') + d^M_{\min}(r',v)$ so if the result holds for r' it holds for r. Without loss of generality assume that there is no such r'. Now consider the following cases:

- If q contains v then $d^M_{\min}(r,v) = d^M_{\min}(r,t) - d^M_{\min}(v,t) < d^M_{\min}(r,t)$.
- If p contains v then $d^M_{\min}(r,v) = d^M_{\min}(r,s) - d^M_{\min}(v,s) < d^M_{\min}(r,s) \leq d^M_{\min}(r,t)$.
- Otherwise v is on some other (induced) path between s and t. But now $r \xrightarrow{p} s \xrightarrow{} v \xrightarrow{} t \xrightarrow{q} r$ forms a cycle of length ≥ 4. Since $\mathcal{M}^+$ is triangulated there must be an edge from v to some node on p or q. There cannot be an edge between s and t or else there would not be any induced paths between s and t. But then $d^M_{\min}(r,v) \leq d^M_{\min}(r,t)$.

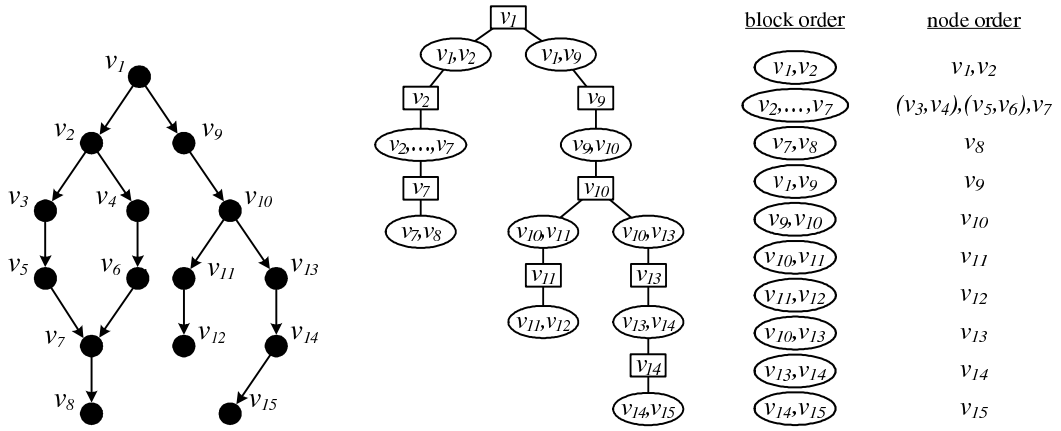


∎

Algorithm 7: Block-Shortest-Path Ordering

Input : $\mathcal{G}$ // input Bayesian network
$\mathcal{M}^+$ // corresponding moralized triangulation
Output: O // an ordering $X_1,\ldots,X_N$

$O \leftarrow \emptyset$
$O_T \leftarrow$ topological ordering of the nodes in $\mathcal{G}$
$O_B \leftarrow$ depth-first search ordering of blocks in $\mathcal{M}^+$
while $O_B \neq \emptyset$ **do**
  $B \leftarrow$ pop O_B
  $R \leftarrow$ cut-vertex of B with lowest O_T
  Push nodes in B to O in order of $(O_T, d^M_{\min}(R,\cdot))$
end
return O



(a) Bayesian Network, $\mathcal{G}$ (b) Block Tree, $\mathcal{B}$ (c) BSP Ordering

Figure 7: Concrete example of BSP ordering using the Bayesian network from Figure 4. Nodes in parentheses are the same distance from the root cut-vertex and can be ordered arbitrarily.

We now use this result to order nodes according to their distance from the cut-vertex in the block that connects it to the blocks already ordered (which we call the root cut-vertex). Algorithm (7) shows how our Block-Shortest-Path (BSP) ordering is constructed and Figure 7 demonstrates the application of that ordering to a concrete example.

Finally, we note that the above ordering, while almost strict, still allows for variables that are the same distance from the root cut-vertex of the block to be ordered arbitrarily. Indeed, as the following example shows two nodes that are the same distance from the block cut-vertex can be symmetrically contaminating. We break such ties arbitrarily.

Example 7.4: Consider, again, the example network shown in Example 7.2. The set of nodes $\{v_2, v_3, v_6, v_7, v_8\}$ are all the adjacent to r and so can be ordered arbitrarily. An edge from v_2 to v_8 (or vice versa) will contaminate v_7. Likewise an edge from v_3 to v_6 (or vice versa) will also contaminate v_7. It turns out that for any ordering of these nodes, it is always possible to add an edge that will contaminate other nodes in the set. This is consistent with the contamination result of Theorem 7.3 since these nodes are all equi-distant from r. ∎

8. Experimental Evaluation

In this section we perform experimental validation of our approach and show that it is beneficial for learning Bayesian networks of bounded treewidth. Specifically, we demonstrate that by making use of global structure modification steps, our approach leads to superior generalization. In order to evaluate our method we compare against two strong baseline approaches.

The first baseline is an improved variant of the thin junction tree approach of Bach and Jordan (2002). We start, as in our method, with a Chow-Liu forest and iteratively add the single best scoring edge. To make the approach as comparable to ours as possible, at each iteration, we triangulate the model using either the maximum cardinality search or minimum fill-in heuristics (see, for example, Koster et al., 2001), as well as using our treewidth friendly triangulation, and take the triangulation that results in a lower treewidth.[5] As in our method, when the treewidth bound is reached, we continue to add edges that improve the model selection score until no such edges can be found that do not also increase the treewidth bound.

The second baseline is an aggressive structure learning approach that combines greedy edge modifications with a TABU list (e.g., Glover and Laguna, 1993) and random moves. This approach is not constrained by a treewidth bound. Comparison to this baseline allows us to evaluate the merit of our method with respect to an unconstrained state-of-the-art search procedure.

We evaluate our method on four real-world data sets that are described below. Where relevant we also compare our results to the results of Chechetka and Guestrin (2008).

8.1 Gene Expression

In our first experiment, we consider a continuous data set based on a study that measures the expression of the baker's yeast genes in 173 experiments (Gasch et al., 2000). In this study, researchers measured the expression of 6152 yeast genes in response to changes in the environmental conditions, resulting in a matrix of 173×6152 measurements. The measurements are real-valued and, in our experiments, we learn sigmoid Bayesian networks using the Bayesian Information Criterion (BIC) (Schwarz, 1978) for model selection. For practical reasons, we consider the fully observed set of 89 genes that participate in general metabolic processes (Met). This is the larger of the two sets used by Elidan et al. (2007), and was chosen since part of the response of the yeast to changes in its environment is in altering the activity levels of different parts of its metabolism. We treat the genes as variables and the experiments as instances so that the learned networks indicate possible regulatory or functional connections between genes (Friedman et al., 2000).

Figure 8 shows test log-loss results for the 89 variable gene expression data set as a function of the treewidth bound. The first obvious phenomenon is that both our method (solid blue squares) and

5. We note that in all of our experiments there was only a small difference between the minimum fill-in and maximum cardinality search heuristics for upper bounding the treewidth of the model at hand.

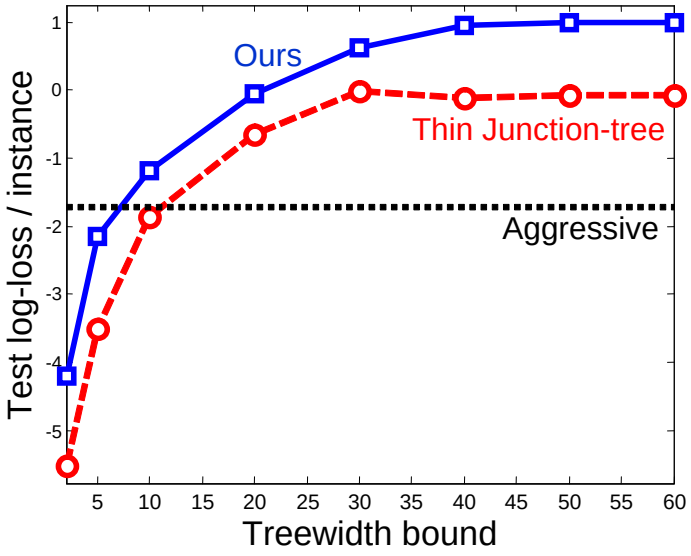


Figure 8: Average test set log-loss per instance over five folds (y-axis) versus the treewidth bound (x-axis) for the 89 variable gene expression data set. Compared are our method (solid blue squares) with the **Thin junction tree** approach (dashed red circles), and an **Aggressive** greedy approach of unbounded treewidth that also uses a TABU list and random moves (dotted black).

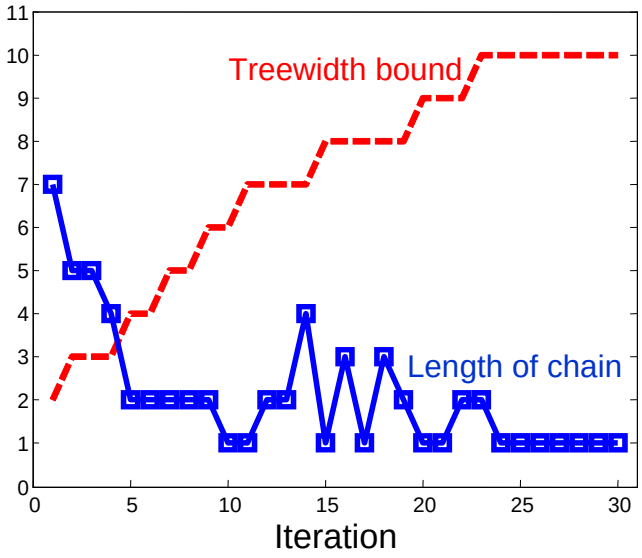


Figure 9: Plot showing the number of edges (in the learned chain) added during each iteration for a typical run with treewidth bound of 10 for the 89 variables gene expression data set. The graph also shows our treewidth estimate at the end of each iteration.

the greedy junction tree approach (dashed red circles) are superior to the aggressive baseline (dotted black). As one might expect, the aggressive baseline achieves a higher BIC score on training data (not shown), but overfits due to the scarcity of the data. By greedy edge addition (the junction tree approach) or global chain addition (our approach), this overfitting is avoided. Indeed, a better choice of edges, that is, ones chosen using a global operator, can lead to increased robustness and better generalization. This is evident by the consistent superiority of our method (solid blue squares) over the greedy variant (dashed red circles). Importantly, even when the treewidth bound is increased passed the saturation point our method surpasses both the thin junction tree approach of Bach and Jordan (2002) and the aggressive search strategy. In this case, we are learning unbounded Bayesian networks and all of the benefit comes from the global nature of our structure modifications.

To qualitatively illustrate the progression of our algorithm from iteration to iteration, we plot the number of edges in the chain (solid blue squares) and treewidth estimate (dashed red) at the end of each iteration. Figure 9 shows a typical run for the 89 variable gene expression data set with treewidth bound set to 10. Our algorithm aggressively adds many edges (making up an optimal chain) per iteration until parts of the network reach the treewidth bound. At that point (iteration 24) the algorithm resorts to adding the single best edge per iteration until no more edges can be added without increasing the treewidth (or that have an adverse effect on the score). To appreciate the non-triviality of some of the chains learned with 4, 5 or 7 edges, we recall that the example shows edges added *after* a Chow-Liu model was initially learned. It is also worth noting that despite their complexity, some chains do not increase the treewidth estimate and for a given treewidth bound K, we typically have more than K iterations (in this example 24 chains are added before reaching the treewidth bound). The number of such iterations is still polynomially bounded as for a Bayesian network with N variables adding more than $K \cdot N$ edges will necessarily result in a treewidth that is greater than K.

In order to verify the efficiency of our method we measured the running time of our algorithm as a function of treewidth bound. Figure 10 shows results for the 89 variable gene expression data set. Observe that our method (solid blue squares) and the greedy thin junction tree approach (dashed red circles) are both approximately linear in the treewidth bound. Appealingly, the additional computation required by our method is not significant and the differences between the two approaches are at most 25%. This should not come as a surprise since the bulk of the time is spent on the collection of sufficient statistics from the data.

It is also worthwhile to discuss the range of treewidths considered in the above experiment as well as the Haplotype sequence experiment considered below. While treewidths of 30 and beyond may seem excessive for exact inference, state-of-the-art exact inference techniques (e.g., Darwiche, 2001; Marinescu and Dechter, 2005) can often handle inference in such networks (for some examples see Bilmes and Dechter, 2006). Since, as shown in Figure 8, it is beneficial to learn models with large treewidth, methods such as ours for learning and the state-of-the-art techniques for inference allow practitioners to push the envelope of the complexity of models learned for real applications.

8.2 The Traffic and Temperature Data Sets

We now compare our method to the mutual-information based LPACJT method for learning bounded treewidth model of Chechetka and Guestrin (2008) (we compare to better of the variants presented in that work). While providing theoretical guarantees (under some assumptions), their method is exponential in the treewidth and cannot be used in a setting similar to the gene expression experi-

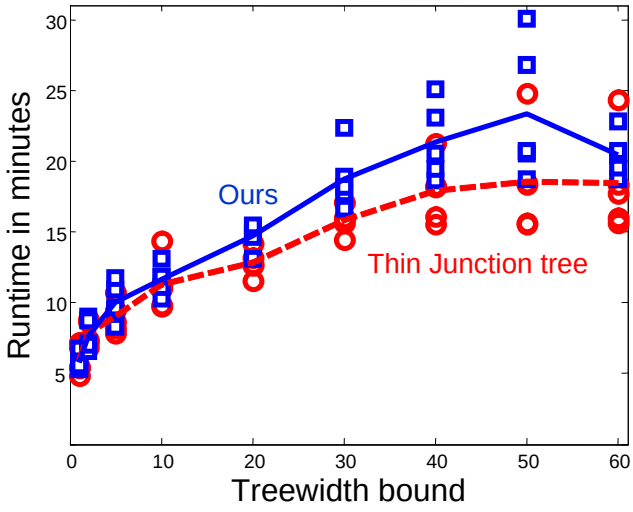
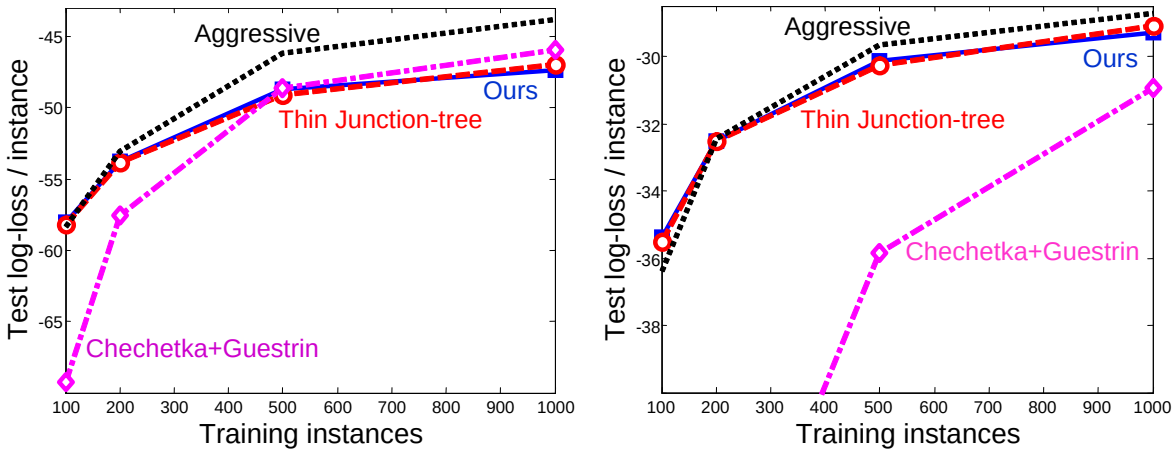


Figure 10: Running time in minutes on the 89 variable gene expression data set (y-axis) as a function of treewidth bound (x-axis). The graph compares our method (solid blue squares) with the thin junction tree approach (dashed red circles). The markers show times for the 5 different fold runs for each treewidth while the line shows the average running time.

ment above. Instead, we compare on the two discrete real-life data set considered in Chechetka and Guestrin (2008). The temperature data is from a two-month deployment of 54 sensor nodes (15K data points) (Deshpande et al., 2004) where each variable was discretized into 4 bins. The traffic data set contains traffic flow information measured every five minutes in 32 locations in California for one month (Krause and Guestrin, 2005). Values were discretized into 4 bins. For both data sets, to make the comparison fair, we used the same discretization and train/test splits as in Chechetka and Guestrin (2008). Furthermore, as their method can only be applied to a small treewidth bound, we also limited our model to a treewidth of two. Figure 11 compares the different methods. Both our method and the thin junction tree approach significantly outperform the LPACJT on small sample size. This result is consistent with that reported in Chechetka and Guestrin (2008) and is due to the fact that the LPACJT method does not naturally use regularization which is crucial in the sparse-data regime. The performance of our method is comparable to the greedy thin junction tree approach with no obvious superiority to either method. This should not come as a surprise since the fact that the unbounded aggressive approach is not significantly better suggests that the strong signal in the data can be captured rather easily. In fact, Chechetka and Guestrin (2008) show that even a Chow-Liu tree does rather well on these data sets (compare this to the gene expression data set where the aggressive variant was superior even at a treewidth of four).

8.3 Haplotype Sequences

Finally we consider a more difficult discrete data set consisting of a sequence of binary single nucleotide polymorphism (SNP) alleles from the Human HapMap project (Consortium, 2003). Our



(a) Temperature (54 variables)

(b) Traffic (32 variables)

Figure 11: Average test set log-loss per instance over five folds (y-axis) versus the number of training instances (x-axis) for the temperature and traffic data sets. Compared are our method (solid blue squares), the **Thin junction tree** approach (dashed red circles), an **Aggressive** greedy approach of unbounded treewidth that also uses a TABU list and random moves (dotted black), and the mutual-information based method of Chechetka and Guestrin (2008) (dash-dot magenta diamonds). For all of our methods except the unbounded **Aggressive**, the treewidth bound was set to two.

model is defined over 200 SNPs (variables) from chromosome 22 of a European population consisting of 60 individuals.[6] In this case, there is a natural ordering of variables that corresponds to the position of the SNPs in the DNA sequence. Figure 12 shows test log-loss results when this ordering is enforced (thicker lines) and when it is not (thinner) lines. Our small benefit over the greedy thin junction tree approach of Bach and Jordan (2002) when the treewidth bound is non-trivial (>2) grows significantly when we take advantage of the natural variable order. Interestingly, this same order decreases the performance of the thin junction tree method. This should not come as a surprise as the greedy method does not make use of a node ordering, while our method provides optimality guarantees with respect to a variable ordering at each iteration. Whether constrained to the natural variable ordering or not, our method ultimately also surpasses the performance of the aggressive unbounded search approach.

9. Discussion and Future Work

In this work we presented a novel method for learning Bayesian networks of bounded treewidth in time that is polynomial in *both* the number of variables and the treewidth bound. Our method builds on an edge update algorithm that dynamically maintains a valid moralized triangulation in a way that facilitates the addition of chains that are guaranteed to increase the treewidth by at most one. We demonstrated the effectiveness of our treewidth-friendly method on real-life data sets, and

6. We considered several different sequences along the chromosome with similar results.

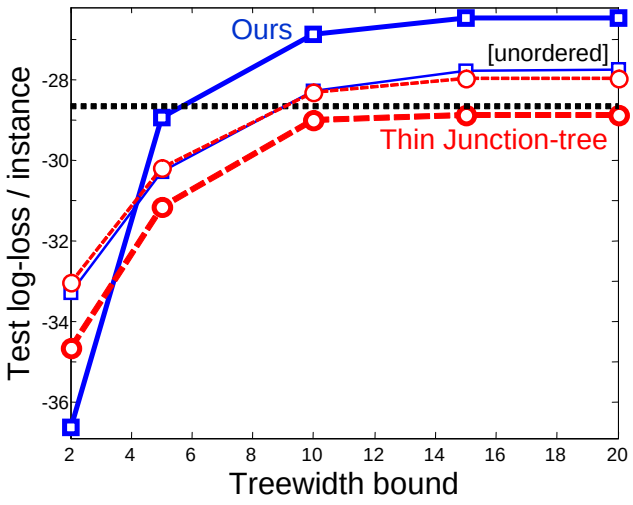


Figure 12: Average test set log-loss per data instance over five folds (y-axis) versus the treewidth bound (x-axis) for the 200 variable Hapmap data set. The graph compares our method (solid blue squares) with the greedy approach (dashed red circles), and an aggressive greedy approach of unbounded treewidth that also uses a TABU list and random moves (dotted black). The thicker lines show the results for a fixed ordering of the variables according to the location along the DNA sequence. The thinner lines show the results without any constraint on the node ordering.

showed that by using global structure modification operators, we are able to learn better models than competing methods even when the treewidth of the models learned is not constrained.

Our method can be viewed as a generalization of the work of Chow and Liu (1968) that is constrained to a chain structure but that provides an optimality guarantee (with respect to a node ordering) at every treewidth. In addition, unlike the thin junction trees approach of Bach and Jordan (2002), we also provide a guarantee that our estimate of the treewidth bound will not increase by more than one at each iteration. Furthermore, we add multiple edges at each iteration, which in turn allows us to better cope with the problem of local maxima in the search. To our knowledge, ours is the first method for efficiently learning bounded treewidth Bayesian networks with structure modifications that are not fully greedy.

Several other methods aim to generalize the work of Chow and Liu (1968). Karger and Srebro (2001) propose a method that is guaranteed to learn a good approximation of the optimal Markov network given a treewidth bound. Their method builds on a hyper-graph that is exponential in the treewidth bound. Chechetka and Guestrin (2008) also propose an innovative method with theoretical guarantees on the quality of the learned model (given some mild assumptions on the generating distribution), but in the context of Bayesian networks. However, like the approach of Karger and Srebro (2001), the method is exponential in the treewidth bound. Thus, both approaches are only practical for treewidths that are much smaller than the ones we consider in this work. In addition, the work of Chechetka and Guestrin (2008) does not naturally allow for the use of regularization.

This has significant impact on performance when the number of training samples is limited, as demonstrated in Section 8.

Meila and Jordan (2000) suggested the use of a mixture of trees, generalizing the Chow-Liu tree on an axis that is orthogonal to a more complex Bayesian network. They provide an efficient method for obtaining a (penalized) likelihood local maxima but their work is limited to a particular and relatively simple structure. Dasgupta (1999) suggested the use of poly-trees but proved that learning the optimal poly-tree is computationally difficult. Other works study this question but in the context where the *true distribution* is assumed to have bounded treewidth (e.g., Beygelzimer and Rish, 2004; Abbeel et al., 2006, and references within).

Our method motivates several exciting future directions. It would be interesting to see to what extent we could overcome the limitation of having to commit to a specific node ordering at each iteration. While we provably cannot consider any node ordering, it may be possible to polynomially provide a reasonable approximation. Second, it may be possible to refine our characterization of the contamination that results from an edge update, which in turn may facilitate the addition of more complex treewidth-friendly structures at each iteration. Finally, we are most interested in exploring whether tools similar to the ones employed in this work could be used to dynamically update the bounded treewidth structure that is the approximating distribution in a variational approximate inference setting.

Acknowledgments

We are grateful to Ben Packer for many useful discussions and comments. Much of the current work was carried out while Gal Elidan was in Stanford University.

References

P. Abbeel, D. Koller, and A. Y. Ng. Learning factor graphs in polynomial time and sample complexity. *Journal of Machine Learning Research*, 7:1743–1788, 2006.

F. Bach and M. I. Jordan. Thin junction trees. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, Mass., 2002. MIT Press.

A. Beygelzimer and I. Rish. Approximability of probability distributions. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

J. Bilmes and R. Dechter. Evaluation of probabilistic inference, the twenty second conference on uncertainty in artificial intelligence. `ssli.ee.washington.edu/∼bilmes/UAI06InferenceEvaluation`, 2006.

H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on 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*, pages 273–280. MIT Press, Cambridge, MA, 2008.

D. M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H. J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer-Verlag, New York, 1996.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. on Info. Theory*, 14:462–467, 1968.

The International HapMap Consortium. The international hapmap project. *Nature*, 426:789–796, 2003.

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

P. Dagum and M. Luby. An optimal approximation algorithm for baysian inference. *Artificial Intelligence*, 60:141–153, 1993.

A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126, 2001.

S. Dasgupta. Learning polytrees. In K. Laskey and H. Prade, editors, *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)*, pages 134–141, San Francisco, 1999. Morgan Kaufmann.

A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Very Large Data Bases (VLDB) Conference*, 2004.

R. Diestel. *Graph Theory*. Springer, 3rd edition, 2005.

G. A. Dirac. On rigid circuit graphs. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 25, Universität Hamburg, 1961.

G. Elidan, I. Nachman, and N. Friedman. "ideal parent" structure learning for continuous variable bayesian networks. *Journal of Machine Learning Research*, 8:1799–1833, 2007.

N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *Computational Biology*, 7:601–620, 2000.

A. Gasch, P. Spellman, C. Kao, O. Carmel-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown. Genomic expression program in the response of yeast cells to environmental changes. *Molecular Biology of the Cell*, 11:4241–4257, 2000.

F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

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

D. Karger and N. Srebro. Learning markov networks: maximum bounded tree-width graphs. In *Symposium on Discrete Algorithms*, pages 392–401, 2001.

A. Koster, H. Bodlaender, and S. Van Hoesel. Treewidth: Computational experiments. Technical report, Universiteit Utrecht, 2001.

A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In F. Bacchus and T. Jaakkola, editors, *Proc. Twenty First Conference on Uncertainty in Artificial Intelligence (UAI '05)*, San Francisco, 2005. Morgan Kaufmann.

W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

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

R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. *IJCAI*, 2005.

C. Meek. Finding a path is harder than finding a tree. *Journal of Artificial Intelligence Research*, 15:383–389, 2001.

M. Meila and M. I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.

M. Narasimhan and J. Bilmes. Pac-learning bounded tree-width graphical models. In M. Chickering and J. Halpern, editors, *Proc. Twenieth Conference on Uncertainty in Artificial Intelligence (UAI '04)*, San Francisco, 2003. Morgan Kaufmann.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1987.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

R. Tarjan and M. Yannakakis. Simple linear time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:3:566–579, 1984.