Learning Bounded Tree-width Bayesian Networks using Integer Linear Programming

Pekka Parviainen
Royal Institute of Technology
Science for Life Laboratory
Swedish e-Science Research Center
Stockholm, Sweden

Hossein Shahrabi Farahani
University of British Columbia
Department of Pathology
Vancouver, BC, Canada

Jens Lagergren
Royal Institute of Technology
Science for Life Laboratory
Swedish e-Science Research Center
Stockholm, Sweden

Abstract

In many applications one wants to compute conditional probabilities given a Bayesian network. This inference problem is NP-hard in general but becomes tractable when the network has low tree-width. Since the inference problem is common in many application areas, we provide a practical algorithm for learning bounded tree-width Bayesian networks. We cast this problem as an integer linear program (ILP). The program can be solved by an anytime algorithm which provides upper bounds to assess the quality of the found solutions. A key component of our program is a novel integer linear formulation for bounding tree-width of a graph. Our tests clearly indicate that our approach works in practice, as our implementation was able to find an optimal or nearly optimal network for most of the data sets.

1 INTRODUCTION

A *Bayesian network* is a representation of a joint probability distribution. It consists of a *structure* and *parameters*. The structure is represented by a *directed acyclic graph (DAG)* that expresses the conditional dependencies and independencies between variables. Parameters, on the other, determine conditional probability distributions associated with each variable.

One common task with Bayesian networks is *inference*, that is, computing conditional probabilities of some variables given some other variables. The inference problem is known to be NP-hard for both the exact (Cooper, 1990) and the approximate (Dagum and Luby, 1993) variant. Inference is, however, tractable if the network has bounded *tree-width*[1]. Actually, bounding tree-width is necessary for tractable inference; bounding any other property of the network keeps inference intractable in the worst-case (Chandrasekaran et al., 2008; Kwisthout et al., 2010). Thus, if an application requires that one has to be able to infer fast it is necessary that the network has bounded tree-width.

The Bayesian network is not always given. However, if one has access to samples from the distribution one can learn the network from data. Learning the structure of a Bayesian network is NP-hard (Chickering, 1996; Chickering et al., 2004). Despite of this, exact algorithms have been developed actively in recent years. Approaches have included dynamic programming (Koivisto and Sood, 2004; Silander and Myllymäki, 2006; Parviainen and Koivisto, 2009; Malone et al., 2011) and branch-and-bound (de Campos et al., 2009). Maybe the most promising approach so far is integer linear programming. The state-of-the-art integer linear programming algorithms have been able to find provably optimal networks for data sets of over 100 variables (Jaakkola et al., 2010; Cussens, 2011; Cussens and Bartlett, 2013).

To ensure that it is possible to infer fast in the learned network, one can learn a Bayesian network with bounded tree-width. However, this does not come without drawbacks. To be able to represent a joint probability distribution exactly, a Bayesian network has to express all the dependencies in the underlying distribution. Arcs in a Bayesian network encode de-

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

[1]Here the bounded tree-width means that tree-width is bounded by some small constant. In this case the time requirement for inference grows linearly with the number of nodes.

pendencies between random variables and sometimes dependencies cannot be represented by a low tree-width network. Thus, forcing a Bayesian network to have a bounded tree-width often makes it impossible to represent the underlying distribution exactly. However, in some applications it may be beneficial to trade exactness of the representation of the distribution for efficiency of inference. In this case, the modeler should use the largest tree-width bound that renders inference fast enough for the application in question.

Research on learning bounded tree-width Bayesian networks has been limited so far. It has been known for a long time that trees, that is, networks with tree-width one, can be learned in polynomial time (Chow and Liu, 1968). Although there are several publications on related graphical models (Karger and Srebro, 2001; Srebro, 2001; Bach and Jordan, 2002; Chechetka and Guestrin, 2007; Kumar and Bach, 2013) we are aware of only two publications concerning learning bounded tree-width Bayesian networks in general. First, Elidan and Gould (2008) have published a heuristic algorithm that is fast but gives no guarantees on the quality of the output. Second, recently Korhonen and Parviainen (2013) introduced a dynamic programming algorithm that is guaranteed to find an optimal network. However, their algorithm is extremely slow and of little practical use.

Motivated by the need for fast inference and the deficiencies of the existing algorithms, we present a new algorithm for learning bounded tree-width Bayesian networks from data. We take the so-called score-based approach (Cooper and Herskovits, 1992; Heckerman et al., 1995) which is based on assigning a score to each structure, depending on how well it fits to the data. The score-based approach makes it easy to compare "goodness" of different structures even when the structures do not express all the dependencies among variables. This makes the score-based approach a natural choice in constrained structure learning, such as the present case.

Learning the structure of a bounded tree-width Bayesian network is known to be an NP-hard problem (Korhonen and Parviainen, 2013). Thus, there is little hope for exact polynomial time algorithms. Further, the results of Korhonen and Parviainen (2013) suggest that adding the requirement of bounded tree-width makes learning of Bayesian networks more complex and time-consuming than in the case of unbounded tree-width Bayesian networks. Encouraged by the earlier results on using integer linear programming to learn Bayesian networks, we formulate the problem of learning bounded tree-width Bayesian network as an integer linear program (ILP). The objective function corresponds to the score of the network. Our ILP has two sets of constraints: one set guarantees that the structure is a valid DAG and another that its tree-width is below a given threshold. This produces an anytime algorithm: If the algorithm is let run long enough, it will always return an optimal solution. On the other hand, if it is aborted earlier, it will return a feasible solution, that is, some DAG with bounded tree-width. The algorithm also provides an upper bound for the score, making it possible to assess the quality of the solution.

We continue by introducing the necessary preliminaries on Bayesian networks and tree-width. Then in Section 3 we present the theoretical results behind our ILP. We formulate the ILP in Section 4. We proceed to present practical details and performance results in Section 5. Finally we discuss some future directions in Section 6.

2 PRELIMINARIES

2.1 Bayesian Networks

In this paper we are interested in the structure of a Bayesian network. The structure of a Bayesian network is represented by a *directed acyclic graph (DAG)*. A DAG is denoted by (N, A), where N is the node set and A is the arc set. If there is an arc from node u to v we say that u is a *parent* of v and v is a *child* of u. The set of the parents and the children of v in A are denoted by A_v and A^v, respectively. The cardinality of N is denoted by n. If there is no ambiguity about the node set we identify a DAG by its arc set A. A complete directed graph is called a *tournament*.

In this paper, we consider the score-based approach to structure discovery in Bayesian networks. In this approach each DAG is assigned a score based on how well it fits to the data. We are particularly interested in decomposable scores, where the score of a DAG is a sum of local scores. Formally, the score of a DAG A is defined to be

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

where local scores $f(v, A_v)$ are computed from the data. We note that this approach subsumes many commonly used scores like BIC and BDeu.

Usually it is necessary to restrict the number of potential parent sets. To this end, we denote a family of potential parent sets for node v by $\mathcal{F}_v$. By convention $f(v, S) = -\infty$ if $S \notin \mathcal{F}_v$. Commonly, the modeler sets a maximum size for parent sets. Note that the number of potential parent sets can be reduced also by various pruning methods; see, for example, de Campos and Ji (2011).

2.2 Tree-width

An undirected graph is denoted by $G = (V, E)$, where V is the vertex set and E is the edge set.

A *tree decomposition* of G is a pair (X, T), where $X = \{X_1, X_2, \ldots, X_m\}$ is a collection of subsets of V and T is a tree on X, such that

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

The *width* of a tree decomposition is defined as $\max |X_i| - 1$. The *tree-width of an undirected graph* G is the minimum width over all tree decompositions of G.

A *skeleton* of a directed graph (N, A) is an undirected graph (N, E), where an edge $\{u, v\}$ is included in E if and only if there is an arc uv in A. A *moralized graph* of a directed acyclic graph (N, A) is an undirected graph (N, E), where an edge $\{u, v\}$ is included in E if and only if there is an arc uv in A or there are arcs us and vs in A for some s. The *tree-width of a DAG* is defined as the tree-width of its moralized graph (Elidan and Gould, 2008). It should be noted that a node and its parents form a clique in the moralized graph. Thus, the number of parents lower bounds the tree-width.

There are several equivalent characterizations for tree-width. For our purposes, it is useful to consider the elimination ordering formulation. To this end, let us consider triangulated graphs. An undirected graph is called *triangulated* (or *chordalized*) if all cycles of length 4 or more have a *chord*, that is, an edge between two non-adjacent vertices. Every graph can be transformed into a triangulated graph by adding edges. The *width* of a triangulated graph is the size of its largest clique minus one. The tree-width of a graph equals to the minimum width over all of its triangulations.

A graph can be triangulated using an elimination ordering formulation. It is well known that, given an ordering of vertices, a graph can be triangulated by going through all vertices in that order and for each vertex adding an edge between its higher numbered neighbors; these edges are called *fill-in edges*. It is also known that if a graph is triangulated then there exists a *perfect elimination order*, that is, an order such that the above procedure does not add any edges. Computing the width of a triangulated graph given a perfect elimination order is easy: it is the maximum number of higher numbered neighbors over all vertices.

A *k-tree* is defined as follows. A clique of $k+1$ vertices is a k-tree. Given a k-tree, a new k-tree can be formed by adding a new node adjacent to exactly k vertices that form a clique in the original k-tree. It should be noted that k-trees are maximal graphs of tree-width k, that is, every graph with tree-width at most k is a subgraph of at least one k-tree. A k-tree with n vertices has $(n-k)k + \binom{k}{2}$ edges.

2.3 Problem Statement

Our goal is to find a Bayesian network structure that has bounded tree-width and maximizes the score.

Problem 1 *Given local scores $f(v, S)$ for all $v \in N$ and $S \in \mathcal{F}_v$ and a parameter k, find a DAG A such that the score $f(A)$ is maximized and tree-width of A is at most k.*

3 BOUNDING TREE-WIDTH

On a high level, it is easy to represent Problem 1 as a constrained optimization problem: One maximizes the score of a directed graph under constraints that the graph must be acyclic and tree-width of its moralized graph is at most k. The key question is how to formulate the constraints so that the program is fast to solve.

One straightforward way to formulate Problem 1 as an integer linear program would be to take existing formulations for acyclicity and tree-width and combine them. Initially, we merged the ILPs for learning (unbounded tree-width) Bayesian networks by Cussens (2011) and computing tree-width of an undirected graph by an elimination ordering formulation (Grigoriev et al., 2011). However, we were not satisfied with the results of our preliminary tests with this approach. Therefore, we present another approach controlling tree-width.

We work with two directed graphs that are represented by sets of binary variables. The first set of variables, z_{Sv} for all $v \in N$ and $S \in \mathcal{F}_v$, encodes the structure of the Bayesian network. In other words, z_{Sv} takes value 1 when the parent set of v is S; the graph implied by this encoding is called the z-graph. For guaranteeing the acyclicity of the z-graph, we use cluster constraints by Cussens (2011). To guarantee that the z-variables encode a graph with tree-width at most k, we introduce an auxiliary variable set for controlling tree-width. The y-variables, y_{ij} are defined, for technical reasons explained later in this section, for $i, j \in N'$, where N' is a superset of N. The y-variables encode a directed graph, called the y-graph, whose skeleton is a k-tree such that the moralized graph of the z-graph is its subgraph.

Now the question is what kind of conditions are needed to guarantee that the y-graph is a k-tree. To this end, we introduce a new graph that we call a root graph.

Let N be the node set of the original problem. We notice that in an acyclic graph whose skeleton is a k-tree there is, for each $i \in \{0, 1, \ldots, k-1\}$, exactly one node with i children. This kind of constraints are difficult to express as linear constraints. To circumvent this problem, we introduce a set R of auxiliary variables; we call members of R as roots. A *root graph* $G = (N \cup R, A)$ is a directed graph such that

1. there are no directed cycles of length 3 in G,
2. for each vertex $v \in N$ there are k children that form a tournament,
3. the size of R is k, the nodes in R form a tournament and the nodes in R do not have children outside R,
4. The induced subgraph $G[N]$ has exactly $(n-k)k+\binom{k}{2}$ arcs, and
5. for each set $W \subseteq N$, $|W| \geq k$ it holds that induced subgraph $G[W]$ has at most $(|W|-k)k+\binom{k}{2}$ arcs.

Further, a *semi-root graph* $H = (N \cup R, A)$ is a directed graph such that the conditions (1)–(4) hold.

Next, we will show that induced subgraph $G[N]$ is a k-tree. To do that, we need to show that a root graph G has no directed cycles. To this end, we start by proving couple of results that will be needed to prove the acyclicity. An arc uv is called a *prime arc* if v has no parents in the children of u. Note that because of conditions (1) and (2), if a node in a semi-root graph has an outgoing arc it also has an outgoing prime arc. Notice also that if uv is a prime arc then $A^u \setminus \{v\} \subseteq A^v$. A directed cycle $C = (v_1 v_2 \ldots v_l)$ is called a *prime cycle* if each arc between consecutive nodes is a prime arc.

Next, we will prove a result that are used in a later proofs.

Lemma 1 *For a prime cycle $C = (v_1 v_2 \ldots v_l)$,*

$$\left(\bigcup_{1 \leq i \leq l} A^{v_i} \right) \setminus \{v_1, v_2, \ldots, v_l\} \subseteq A^{v_l}.$$

Proof. Notice that since C is a prime cycle, for each $1 \leq i \leq l-1$ it holds that $A^{v_i} \setminus \{v_{i+1}\} \subseteq A^{v_{i+1}}$ and hence it follows inductively that, for each $1 \leq j \leq l$,

$$\left(\bigcup_{1 \leq i \leq j} A^{v_i} \right) \setminus \{v_1, v_2, \ldots, v_j\} \subseteq A^{v_j}.$$

We conclude the proof. □

Armed with the previous lemma, we will move to show that if a semi-root graph has a prime cycle of length 4 or more then there exists an induced subgraph that has "too many" arcs. We will also show that if there is a cycle in a semi-root graph then there is a prime cycle.

Lemma 2 *Assume that $C = (v_1 v_2 \ldots v_l)$ is a prime cycle on a semi-root graph $G = (N \cup R, A)$. Let $K = A^{v_l} \setminus \{v_1, v_2, \ldots, v_l\}$ and $G' = G[K \cup \{v_1, v_2, \ldots, v_l\}]$. Let n' and m' be the number of nodes and arcs, respectively, of G'. Then*

$$m' > (n'-k)k + \binom{k}{2}.$$

Proof. Notice that by the construction of G, C contains no roots and hence $|A^{v_i}| = k$ for $i \in [l]$. Also it follows from Lemma 1 that A^{v_i} belongs to G' for each $i \in [l]$. Hence, G' has lk outgoing arcs from $\{v_1, v_2, \ldots, v_l\}$. Moreover, $K \subseteq A^{v_l}$ and thus $G'[K]$ is a tournament and has $\binom{|K|}{2}$ arcs. So,

$$m' = lk + \binom{|K|}{2}.$$

However, v_l has k children and one of its children, i.e., v_1, belongs to $\{v_1, v_2, \ldots, v_l\}$ and thus $|K| \leq k-1$. Consequently, $l \geq (n'-k)+1$ and

$$lk + \binom{|K|}{2} > (n'-k)k + \binom{k}{2}.$$

The lemma follows. □

Lemma 3 *If a semi-root graph G has a cycle, then it has a prime cycle.*

Proof. Assume that among all cycles, $C = (v_1 v_2 \ldots v_l)$ starts with a maximum number of consecutive prime arcs (from v_1). Assume that C is not a prime cycle, $v_i v_{i+1}$ is the first non-prime arc of C and $v_i u$ is a prime arc.

If $u \notin C$ then $C' = (v_1 v_2 \ldots v_i u v_{i+1} \ldots v_l)$ is a cycle on G and it starts with more prime arcs than C contradicting the assumption that C starts with the maximum number of prime arcs. We conclude that $u \in C$.

Assume $u = v_j$. If $j < i$ then $(v_j \ldots v_i)$ is a prime cycle and if $j > i$ then $(v_1 \ldots v_i v_j v_{j+1} \ldots v_l)$ is a cycle starting with more prime arcs than C contradicting the

assumption that C starts with the maximum number of prime arcs.

We conclude that G has a prime cycle. □

Now we are ready to prove that root graphs are acyclic.

Lemma 4 *If G is a root graph then there are no directed cycles in G.*

Proof. We notice that, by the definitions of a root graph and semi-root graph, G is a semi-root graph such that for each set $W \subseteq N$, $|W| \geq k$ it holds that induced graph $G[W]$ has at most $(|W|-k)k+\binom{k}{2}$ arcs. Thus, by Lemmas 2 and 3, G has no directed cycles. □

Now, the following lemma implies that the skeleton of an induced subgraph $G[N]$ of a root graph is a k-tree.

Lemma 5 *Let $G = (N \cup R, A)$ be a directed graph such that (1) there are no directed cycles, (2) for each vertex $v \in N$ there are k children that form a tournament, (3) the size of R is k, the nodes in R form a tournament and the nodes in R do not have children outside R, and (4) the induced subgraph on N has $(|N|-k)k+\binom{k}{2}$ arcs. Then the skeleton of $G[N]$ is a k-tree.*

Proof. We proof this lemma by induction on the number of nodes.

First, let us proof that if $|N| = k$ then the skeleton of $G[N]$ is a clique. By the definition of G the induced subgraph $G[N]$ has $k(k-1)/2$ arcs, that is, a tournament. It is easy to see that this tournament can be constructed without violating the other constraints. The skeleton of a tournament is a clique and thus the claim follows.

Now, assume that G has $n > k$ nodes and the lemma holds for any smaller graph. Since (1) and (3) hold, there is a node $v \in N$ with no parents. By inductive assumption, the skeleton of $G \setminus \{v\}$ is a k-tree. Moreover, by (2) the children of v are a clique in the skeleton of $G \setminus \{v\}$. We also note that by (2) and (4) all the children of v need to be in N. Thus, by the definition of k-tree, the skeleton of G is a k-tree. □

Lemma 5 implies that to guarantee bounded tree-width it is sufficient that the y-graph is a root graph. Note that the proof of the lemma actually implies that the direction of the arcs in G, in a natural way, induce a perfect elimination order of its skeleton.

4 INTEGER LINEAR PROGRAM

We are now ready to formulate Problem 1 as an integer linear program (ILP). We start by devising the objective function. To this end, recall that we introduced a set of binary variables z_{Sv}, where $v \in N$ and $S \in \mathcal{F}_v$, to represent the directed graph. Variable z_{Sv} takes value 1 if S is the parent set of v and 0 otherwise. We want to maximize the score. Thus, our objective function is

$$\max \sum_{v \in N} \sum_{S \in \mathcal{F}_v} f(v, S) z_{Sv}.$$

To guarantee that variables z_{Sv} represent a DAG with bounded tree-width, we introduced an auxiliary variable set y_{ij} encoding a root graph. We use a shorthand $N' = N \cup R$. The conditions derived in Section 3 translate to the following linear constraints:

$$\sum_{S \in \mathcal{F}_v} z_{Sv} = 1 \quad \forall v \in N, \tag{1}$$

$$\sum_{j \in N' \setminus \{i\}} y_{ij} = k \quad \forall i \in N, \tag{2}$$

$$\sum_{i \in N} \sum_{j \in N \setminus \{i\}} y_{ij} = nk - k(k+1)/2, \tag{3}$$

$$z_{Si} - y_{ij} - y_{ji} \leq 0 \quad \forall i \in N, j \in S, S \in \mathcal{F}_i, \tag{4}$$

$$z_{Sv} - y_{ij} - y_{ji} \leq 0 \quad \forall i, j \in S, v \in N, S \in \mathcal{F}_v, \tag{5}$$

$$y_{ij} + y_{il} - y_{jl} - y_{lj} \leq 1 \quad \forall i, j, l \in N', \tag{6}$$

$$y_{ij} = 0 \quad \forall i \in R, j \in N, \tag{7}$$

$$y_{ij} = 1 \quad \forall i, j \in R, i < j \text{ in lexicogr. order}, \tag{8}$$

$$y_{ij} + y_{ji} \leq 1 \quad \forall i, j \in N', \tag{9}$$

$$y_{ij} + y_{jl} + y_{li} \leq 2 \quad \forall i, j, l \in N', \tag{10}$$

$$\sum_{i \in W} \sum_{S \in \mathcal{F}_i : S \cap W = \emptyset} z_{Si} \geq 1 \quad \forall W \subseteq N, \tag{11}$$

$$\sum_{i \in W} \sum_{j \in W} y_{ij} \leq (|W| - k)k + \binom{k}{2} \quad \forall W \subseteq N, |W| \geq k, \tag{12}$$

$$y_{ij} \in \{0, 1\} \quad \forall i, j \in N', \tag{13}$$

$$z_{Si} \in \{0, 1\} \quad \forall i \in N, S \in \mathcal{F}_i. \tag{14}$$

Here the constraints (1) and (14) guarantee that each node has exactly one parent set. The constraints (11) are the cluster constraints and together with the two previous constraints they guarantee the acyclicity of the z-graph. The constraints (9) guarantee that the y-graph is a simple graph. The constraints (4) and (5) guarantee that the moralized graph of the z-graph is a subgraph of the y-graph. The constraint (6) guarantees that the y-graph the fill-in edges implied by the elimination order are added, that is, the children of each node form a tournament. The constraint (10) guarantees that the y-graph has no 3-cycles. The constraints (7) and (8) introduce the set R and guarantee the desired properties. The constraint (3) guarantees that the induced subgraph of the y-graph on N has as many arcs as a k-tree should have. The constraint (12) guarantees that none of the induced subgraphs of the y-graph has too many arcs. Thus, together with the constraint (13) constraints (3), (7), (8), (10), and (12) guarantee that the y-graph is a root graph. Finally, the constraint (2) guarantees that the graphs have tree-width at most k.

5 SOLVING THE ILP IN PRACTICE

We used CPLEX 12 to solve our integer linear program. CPLEX uses a branch-and-cut algorithm to solve ILP. It starts with solving a relaxation of the original ILP. Then it starts to tighten the relaxation by adding cuts and branching. In the cutting phase the branch-and-cut algorithm adds new constraints (cutting planes) that cut out the current solution without cutting out any solutions that are feasible in the original ILP. In the branching phase the problem is split into two subproblems by fixing values of some variable. To speed up the algorithm it is essential to find "deep" cuts that tighten the relaxation fast. Further, it is essential to keep the branching tree small by pruning the tree whenever one finds good feasible solutions.

To scale up and speed up CPLEX, we added custom algorithms for finding cutting planes and heuristics for finding "good" feasible solutions. The cuts are described in Section 5.1 and the heuristics in Section 5.2. Customized parts were implemented using Python. Our implementation called TWILP is available as open source software[2].

5.1 Sub-IP

One practical problem with our program is that the constraints (11) and (12) consists of one constraint for each node subset, that is, 2^n constraints in total. This is too much to be explicitly included in the program. Therefore, we start by solving a relaxation and add constraints as cutting planes whenever needed. To this end, it is important to find "good" cutting planes. One way to do this is to solve sub-IPs that try to maximize the distance between the plane and the current solution. For cluster constraints (11) we use the sub-IP introduced by Cussens (2011).

To find good cutting planes for the y-variables, we try to maximize the distance between the current solution and a cutting plane in a similar fashion as Cussens (2011). Let $\hat{y}_{ij}$ be the value of y_{ij} in the current solution. Now the distance of a cutting plane for set $W \subseteq N$ from the current solution is

$$\frac{\sum_{i \in W} \sum_{j \in W \setminus \{i\}} \hat{y}_{ij} - |W|k + k(k+1)/2}{\sqrt{\sum_{i \in W} \sum_{j \in W \setminus \{i\}} 1}}.$$

Following Cussens, we ignore the denominator and attempt to maximize the numerator.

To this end, we introduce new binary variables I_i for all $i \in N$ which takes value 1 if and only if $i \in W$ and I_{ij} for all $i \in W$ and $j \in W \setminus \{i\}$ which takes value 1 if and only both i and j are in W. Now our objective function is

$$\max \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \hat{y}_{ij} I_{ij} - k \sum_{i \in N} I_i.$$

To ensure the conditions mentioned above we require the following conditions

$$2I_{ij} - I_i - I_j \leq 0 \quad \forall i \in N, j \in N \setminus \{i\} \quad (15)$$
$$I_i + I_j - I_{ij} \leq 1 \quad \forall i \in N, j \in N \setminus \{i\}. \quad (16)$$

To ensure that the plane is a cutting plane, that is, it cuts the current solution, we require that

$$\sum_{i \in N} \sum_{j \in N \setminus \{i\}} \hat{y}_{ij} I_{ij} - k \sum_{i \in N} I_i > -k(k+1)/2. \quad (17)$$

Finally, we are interested only on sets with more than $k+1$ variables. Thus, we have

$$\sum_{i \in N} I_i > k+1. \quad (18)$$

5.2 Feasibility heuristics

In our preliminary tests, we found out that the cuts often quickly give relatively tight upper bounds. However, the program sometimes failed to find any feasible integer solutions in a reasonable time. To speed up finding good feasible solutions we added a custom

[2] `https://bitbucket.org/twilp/twilp/`

Table 1: ILP's performance on various data sets. n is the number of vertices in each data set, k is the upper bound on the tree-width of the network, and score is the BDe score of the network. The column "Unbounded" consists of scores of optimal unbounded tree-width Bayesian networks. Gap OPT means that the solution is guaranteed to be optimal.

Dataset			$k=2$		$k=3$		Unbounded
Name	Sample size	n	Score	Gap	Score	Gap	Score
WATER	10000	32	-128948.06	0.00	-128806.35	0.00	-128705.65
WATER	1000	32	-13265.71	0.00	-13262.34	OPT	-13262.34
WATER	100	32	-1501.67	0.00	-1500.97	OPT	-1500.97
ADULT	32561	15	-352224.55	OPT	-351481.63	0.00	-351151.29
ALARM	10000	37	-105501.64	OPT	-105327.36	0.01	-105278.25
ALARM	1000	37	-11259.78	0.00	-11243.83	0.00	-11240.35
ALARM	100	37	-1371.12	0.00	-1369.78	0.01	-1349.23
ASIA	10000	8	-22466.40	OPT	-22466.40	OPT	-22466.40
ASIA	1000	8	-2318.65	OPT	-2317.41	OPT	-2317.41
ASIA	100	8	-245.64	OPT	-245.64	OPT	-245.64
CARPO	10000	60	-177016.77	0.01	-175180.51	0.02	-174130.56
CARPO	1000	60	-18100.04	0.02	-18300.05	0.03	-17718.95
CARPO	100	60	-1966.74	0.04	-2045.31	0.10	-1848.37
HAILFINDER	10000	56	-502771.20	0.01	-500480.17	0.01	-497651.87
HAILFINDER	1000	56	-52473.96	OPT	-52714.06	0.01	-52473.25
HAILFINDER	100	56	-6021.27	OPT	-6019.47	OPT	-6019.47
HOUSING	506	14	-3295.40	OPT	-3180.30	OPT	-3080.14
INSURANCE	10000	27	-134977.64	0.01	-135172.85	0.02	-132968.58
INSURANCE	1000	27	-13991.96	0.01	-13919.09	0.00	-13887.35
INSURANCE	100	27	-1694.77	0.00	-1687.68	0.00	-1686.23
KREDIT	NA	18	-16695.67	OPT	-16695.67	OPT	-16695.67

feasibility heuristic that finds a feasible solution given an infeasible one.

The feasibility heuristic starts by taking the solution of current relaxation. Then it constructs a directed graph in two different ways. First, by choosing for each node a parent set with the highest value in the current solution and second by choosing for each node a parent set with the highest local score given the node-parent set variable has a non-zero value in the current solution.

Next, the heuristic considers both directed graphs separately and transforms them into feasible solutions. If the resulting graph is cyclic the feasibility heuristic uses the heuristic by Eades et al. (1993) to find a small feedback arc set to be removed. After the removal of the feedback arc set we have an DAG. Then, the feasibility heuristic uses various heuristics, like minimum fill-in and minimum cardinality, to find an upper bound for tree-width of the DAG. If none of the tree-width heuristics gives an upper bound that is smaller than or equal to the desired tree-width, the feasibility heuristic removes one arc from the DAG and tries the tree-width heuristics again. This is repeated until the solution is guaranteed to be feasible.

After the above procedure is completed the feasibility heuristic has produced two DAGs whose tree-width is at most the desired bound. The feasibility heuristic finishes by choosing the DAG that has higher score.

5.3 Results

We tested our implementation using data sets provided by Cussens (2011). Cussens sampled sets of 100, 1000, and 10000 observations from various Bayesian networks and computed BDe scores for node-parent set pairs. He restricted the maximum number of parents to 3 and further pruned the scores using the method by de Campos et al. (2009). This helped to reduce the number of variables in the ILP. To compare our method to the existing methods for learning bounded tree-width Bayesian networks, we tested our implementation also with the data sets ADULT and HOUSING used by Korhonen and Parviainen (2013).

All experiments were performed on a system dual CPU (AMD Opteron 6220) Supermicro nodes with 64 GBs RAM per node.

We learned Bayesian networks with maximum tree-width 2 and 3. In our tests, we allocated maximum 12

hours of CPU time and 4 Gb of memory to CPLEX for each problem instance. If CPLEX did not find a provably optimal solution before it ran out of either time or memory, we picked the best feasible solution from the solutions pool. We used the default parameters of CPLEX in all the tests; various alternative parameter combinations were tested but while some instances were solved faster the overall performance was not enhanced. The results from these tests are shown in Table 1. For comparison, we present also the scores of optimal networks learned without setting an upper bound for the tree-width. The table shows the score of the best network found and the proportional gap between the score of the best network and the upper bound for the score. More formally,

$$\text{gap} = \frac{|s_{FEAS} - s_{UB}|}{|s_{FEAS}|},$$

where s_{FEAS} is the score of the best feasible integer solution found and s_{UB} is the upper bound for the score. The gap OPT means that a provably optimal network was found. While an optimal network is not always found, the gap is very small in most smaller networks. However, the algorithm seems to struggle with CARPO. Another point to note is that the algorithm seems to often have problems with convergence. That is, even if the algorithm has found an optimal network it takes a very long time to prove it. On the other hand, sometimes the algorithm quickly finds a nearly optimal solution but fails to find to optimum. For example, in the case of WATER with 100 observations and tree-width 2 it took about 23 minutes to find the current solution and the upper bound that give a gap 0.00002. However, the algorithm did not manage to improve either the solution or the upper bound for the next 11 hours and 37 minutes.

It should be noted that in some cases a network with tree-width bound 2 has a higher score than a network with tree-width bound 3 (see, e.g., CARPO with 100 and 1000 observations). This indicates that CPLEX may sometimes get stuck in suboptimal solutions.

6 DISCUSSION

In this paper, we presented an integer linear programming algorithm for learning Bayesian networks with bounded tree-width. The current work raises interesting questions for future research.

Our algorithm scales up to networks that are significantly larger than the networks learned by Korhonen and Parviainen (2013). However, there are still some room for improvement. Especially, even though our algorithm usually finds good upper bounds and feasible solutions quickly, there are issues concerning proving optimality. Another possibility is to consider alternative formulations for tree-width and study whether they are easier to solve.

The second line of research is to study the quality of the results, that is, how well the bounded tree-width structures approximate the true underlying distribution. There has been a small scale study on this topic (Beygelzimer and Rish, 2003) but an extensive investigation is missing. As the main motivation for learning Bayesian networks with bounded tree-width is facilitating exact inference, it is essential to know how close the inferred conditional probabilities are from the true conditional probabilities. If one wants to infer fast, there are two alternative approaches: one can learn a network with bounded tree-width and do exact inference or one can learn a network without constraints on tree-width and do approximate inference. Therefore, an empirical study to characterize the conditions under which each of these approaches is better is needed.

Acknowledgments

The work was supported in part by the Swedish Research Council, grant 2013-4993. Most of the work was done while Hossein Shahrabi Farahani was affiliated with School of Computer Science and Communication and Science for Life Laboratory at Royal Institute of Technology. Authors like to thank James Cussens, Matti Järvisalo and Brandon Malone for valuable discussions.

References

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

A. Beygelzimer and I. Rish. Approximability of probability distributions. In *Advances in Neural Information Processing (NIPS)*, 2003.

V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of Inference in Graphical Models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2008.

A. Chechetka and C. Guestrin. Efficient Principled Learning of Thin Junction Trees. In *Advances in Neural Information Processing 20 (NIPS)*, pages 273–280. MIT Press, 2007.

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

D. M. Chickering, D. Heckerman, and C. Meek. Large-Sample Learning of Bayesian Networks is NP-Hard.

Journal of Machine Learning Research, 5:1287–1330, 2004.

C. K. Chow and C. N. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, IT-14 (3):462–467, 1968.

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

G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9(4):309–347, 1992.

J. Cussens. Bayesian network learning with cutting planes. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 153–160. AUAI Press, 2011.

J. Cussens and M. Bartlett. Advances in Bayesian Network Learning using Integer Programming. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

C. P. de Campos and Q. Ji. Efficient Structure Learning of Bayesian Networks Using Constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

C. P. de Campos, Z. Zeng, and Q. Ji. Structure Learning of Bayesian Networks using Constraints. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 113–120. Omnipress, 2009.

P. Eades, X. Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47(6):319–323, 1993.

G. Elidan and S. Gould. Learning Bounded Treewidth Bayesian Networks. *Journal of Machine Learning Research*, 9(2699-2731):2699–2731, 2008.

A. Grigoriev, H. Ensinck, and N. Usotskaya. Integer linear programming formulations for treewidth. Technical Report RM/11/030, Maastricht University, 2011.

D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243, 1995.

T. Jaakkola, D. Sontag, A. Globerson, and M. Meila. Learning Bayesian Network Structure using LP Relaxations. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 358–365, 2010.

D. Karger and N. Srebro. Learning Markov Networks: Maximum Bounded Tree-Width Graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 392–401, 2001.

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

J. H. Korhonen and P. Parviainen. Exact Learning of Bounded Tree-width Bayesian Networks. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2013.

K. S. S. Kumar and F. Bach. Convex Relaxations for Learning Bounded Treewidth Decomposable Graphs. In *Proceedings of the 30th Internationl Conference on Machine Learning (ICML)*, 2013.

J. H. P. Kwisthout, H. L. Bodlaender, and L. C. van der Gaag. The Necessity of Bounded Treewidth for Efficient Inference in Bayesian Networks. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pages 237–242. IOS Press, 2010.

B. Malone, C. Yuan, E. A. Hansen, and S. Bridges. Improving the Scalability of Optimal Bayesian Network Learning with External-Memory Frontier Breadth-First Branch and Bound Search. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 479–488. AUAI Press, 2011.

P. Parviainen and M. Koivisto. Exact Structure Discovery in Bayesian Networks with Less Space. *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 436–443, 2009.

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

N. Srebro. Maximum Likelihood Bounded Tree-Width Markov Networks. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 504–511, 2001.