Bayesian Networks of Bounded Treewidth: A Performance Analysis

Fabio H. S. Machado
Escola Politécnica
Universidade de São Paulo

Denis D. Mauá
Instituto de Matemática e Estatística
Universidade de São Paulo

Fabio G. Cozman
Escola Politécnica
Universidade de São Paulo

***Abstract*—Bounding the treewidth of Bayesian Networks has been claimed to guarantee polynomial-time inference with little harm to accuracy. However, there has been little empirical evidence to support that claim. In this work we study empirically the effect of bounding treewidth on generalization ability. Our results suggest that adding a constraint to treewidth decreases the model performance on unseen data and makes the corresponding optimization problem more difficult.**

I. Introduction

Bayesian networks are graphical models used for efficiently representing dependency relations and joint probability distribuitions in multivariate uncertainty problems [1]. A Bayesian Network has three main components: a set of random variables, a directed acyclic graph (referred to as the network structure) representing (in)dependences between variables, and a collection of conditional probability values that jointly specify a joint probability distribution over the variables. Specifying a Bayesian Network is a daunting task, and practitioners often resort to automatic methods that "learn" both the structure and the conditional probabilities from data [2].

In this paper, we refer to "learning Bayesian Networks" specifically as the problem of learning a Bayesian Network structure from data; given a Bayesian Network structure, we can learn the parameters (i.e., the conditional probabilities) from a complete data set efficiently. A popular approach for learning Bayesian Networks, called score-based learning, uses a score function that assigns a quality measure to any given structure and searches for the score-optimizer structure within a group of candidate structures [3]. The score functions usually reward data fitness while penalizing the model complexity in a sort of Occam's razor policy: when faced with a choice between two models that equally fit the data, choose the least complex one. Maximizing the fitness of data guides the search towards structures that assign high probability to the observed data set. Penalizing complexity reduces model overfitting and increases the generalization ability.

Most score functions penalize a structure by the (weighted) number of edges in the network. Even though this favors sparse graphs that tend to perform better on unseen data, it does not balance for the overall complexity of the model: sparse networks can represent fairly complex distributions, which can still lead to overfitting when learning from small data sets. There is a second reason why penalizing the network edge density may not suffice. Very often, a Bayesian network is learned so that it can be used for drawing arbitrary inferences such as querying the posterior probability of a hypothesis after evidence is entered or selecting the states of a set of variables so as to maximize its conditional probability [4]. These inferences are all known to be NP-Hard to compute, and there is strong evidence that their complexity is exponential in the network treewidth, which is a measure of tree-likeness of a graph [5]–[7]. Controlling for sparsity does not limit the treewidth of the learned structures, and this can lead to learning models that perform poorly in applications due to poor generalization and the use of approximate inference [8], [9]. It is also very common to limit the number of parents (i.e., the in-degree) of nodes in the search. This again does not avoid generating structures of high treewidth. For example, a squared grid-like structure (such as the one in Figure 1) has maximum in-degree of two, less than $2n$ edges, and treewidth $\sqrt{n}$, where n is the number of nodes.

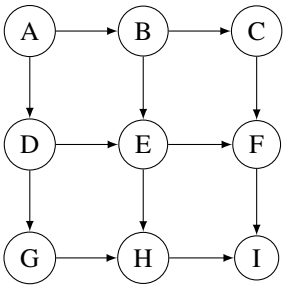


Fig. 1: A directed grid with 9 nodes. Each node in this graph has at most 2 parents. Still, the treewidth is 3.

With these issues in mind, many researchers have suggested constraining the search space, when learning Bayesian Networks, to structures of bounded treewidth, aiming at increased performance of the learned structure in real-world applications [9]–[13]. Since the task of estimating a network's treewidth is itself NP-Hard [14], extending current approaches to learning Bayesian networks in this direction is not trivial, and research on the topic has recently been very intense [9]–[13]. Even though most of these works are motivated by the claim that constraining the search space increases the performance of learned networks, the empirical evidence has not been substantial. Moreover, they have overlooked the additional burden that constraining the treewidth adds to search algorithms. To illustrate this point, note that score functions decompose as a sum of local score functions which depend only on a variable and its (immediate) parents. Hence, once a topological ordering among the variables is fixed, the problem of learning a score-maximizing structure breaks down to a greedy search for the

best parent set of each node; the latter can be solved efficiently if a low bound on in-degree is assumed. This fact is the base of many approaches to learning Bayesian Networks such as the popular Order-Based Sampling method of Teyssier and Koller [15]. Such an approach can no longer be applied if a bound on treewidth is imposed, since the constraint on treewidth makes the subproblems interdependent, and one often has to consider more complex structures (than orderings) such as (partial) k-trees in order to decompose the problem [9], [12].

Our main goal in this paper is to empirically analize the effect that constraining the treewidth has on the generalization ability of learned Bayesian Networks. We do that by applying the mixed-integer linear programming approach to learning bounded treewidth Bayesian Networks described in [9] to a collection of real data sets from the UCI repository. We measure generalization ability by computing the cross-validated data likelihood of the model on held-out data. For each dataset we learn structures with different limits in its treewidth (usually 3, 4, 5 and increments of 5 until the dataset's maximum number of variables). The results suggest that constraining the search space actually leads to more difficult computational problems as the errors returned by programming solvers are often far larger for more constrained problems (ones with a smaller treewidth bound). The severe constraint on treewidth also tends to generate networks which generalize poorly on the test set. When the bound on treewidth is more loose, the performance of the learned networks is competitive with the unconstrained networks.

The remainder of this paper is structured as follows. We start with background knowledge on graph theory and Bayesian Networks (Section II), and then move to the presentation of the mixed-integer linear programming approach to learning bounded treewidth networks that we use (Section III). The experimental results, our main contribution, appear in Section IV, along with a discussion. Conclusions and comments on future work appear in Section V.

II. Preliminaries

In this section we present some necessary background on graph theory and Bayesian Networks.

A. Graph Theory

A Directed Acyclic Graph (DAG) is a graph where the edges have a direction associated with them such that there are no directed cycles (i.e., one cannot reach a node from itself following the direction of the edges). The parent set of a node i, denoted by Pa_i, is the set of all nodes that have an edge directed to this node. Similarly, the children set of a node is the set of all node that it points to. The descendants of a node comprises all the nodes that can be reached from it, the node's children, their children's children, and so on. The set of non-descendants of a node is the complement of its descendants.

The moral graph is the equivalent undirected form of a DAG and it is used by most operations in graphical models. It can be obtained by adding an edge between all pair of nodes that have a common child and dropping edge directions. Figure 2 show the moral graph of Figure 1.

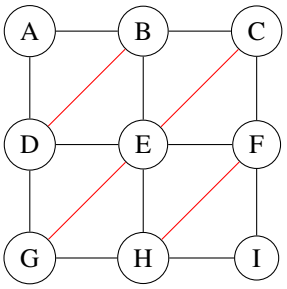
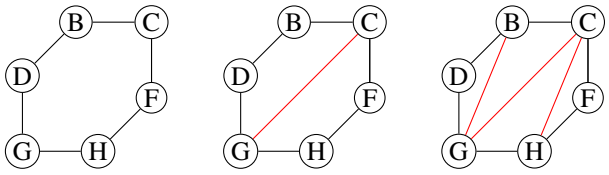


Fig. 2: The moral graph of the graph shown in 1. Red edges are the added edges by linking common parents.



(a) A Cycle (b) New Chord (c) New Chords

Fig. 3: The cicle B-D-G-H-F-C (3a) has no chord. This can be solved by inserting and edge, for example, from node G to C (3b). However this creates tow new chordless cicles G-D-B-C and G-H-F-C. Again, two new edges are inserted (3c).

The treewidth of a DAG is the treewidth of its corresponding moral graph. The treewidth of any undirected graph can be found in the size of the cliques in a chordal graph that contains it (i.e. this graph is a subgraph of the given chrodal graph). A chordal graph is a graph where all cycles of length four or more have a chord. A cycle in an undirected graph has a chord if it contains two nodes that are connected by an edge not in the cycle. Any undirected graph can be made chordal by inserting edges in it, a process called chordalization. A clique is a subset of vertices in an undirected graph that form a complete subgraph (i.e., all vertices are pairwise connected). Figure 3 shows a chordalization of the graph of Figure 2.

The treewidth of a undirected graph can be found in the maximum number of higher ordered neighbors according to its perfect elimination order. A perfect elimination order is a linear ordering of the nodes such that the higher ordered neighbors of each node form a clique. A graph can have a perfect elimination order if and only if it is chordal. Figure 4 shows a chordal graph obtained from Figure 2 and its corresponding perfect elimination order.

Thus, the treewidth of an undirected graph G is the minimum $w \geq 0$ such that G is a subgraph of a chrodal graph with all cliques of size at most $w+1$ [16]. Alternatively, the treewdith of a graph can be computed as the minimum treewidth of a perfect elimination order. The perfect elimination order in Figure 4 has minimum treewidth 3, which is thus the treewidth of the DAG shown in Figure 1.

B. Bayesian Networks

Bayesian Networks are probabilistic graphical models that encode (in)dependences and joint probability distribuitions on

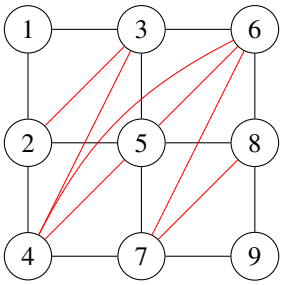
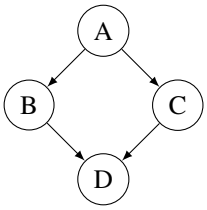


Fig. 4: A perfect elimination order for a chordal graph.



(a) Structure G

$\theta_A(T) = P(A = T) = 0.1$
$\theta_C(T, T) = P(C = T|A = T) = 0.7$
$\theta_C(T, F) = P(C = T|A = F) = 0.2$
$\theta_B(T, T) = P(B = T|A = T) = 0.7$
$\theta_B(T, F) = P(B = T|A = F) = 0.2$
$\theta_D(T, T, T) = P(D = T|C = T, B = T) = 0.9$
$\theta_D(T, T, F) = P(D = T|C = T, B = F) = 0.7$
$\theta_D(T, F, T) = P(D = T|C = F, B = T) = 0.7$
$\theta_D(T, F, F) = P(D = T|C = F, B = F) = 0$

(b) Parameters Θ

Fig. 5: Example of a Bayesian Network.

multivariate domains. Formally a Bayesian Network is a triple $\{V, G, \Theta\}$, where $V = \{X_1, X_2, ..., X_n\}$ is a finite set of random variables; $G = \{V, E\}$ is a DAG where the edge in E represents dependency relationships between variables in V; and $\Theta = \{\theta_i(x_i, x_{Pa_i}) = P(x_i|x_{Pa_i})\}$ is a collection of conditional probability values, one for each value x_i of variable X_i and configuration x_{Pa_i} of its parents Pa_i. A variable X_i is assumed independent of all its non-descendants given its parents Pa_i. Because of this, a Bayesian Network specifies a joint probability distribution over V as:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa_i) = \prod_{i=1}^{n} \theta_i(x_i, x_{Pa_i}). \quad (1)$$

To learn a Bayesian Network is to estimate Θ and G given a dataset D of historical observations of variables V. A possible way to estimate the structure G is to find, for each variable, another (minimal) set of variables that makes it conditionally independet of all the others. This is called constraint-based structure learning; we will not pursue such an approach here. Alternatively, one can formulate the problem as a combinatorial optimization where one searches for a high scoring structure among the set of all possible DAGs. Depending on the choice of scoring function, exact solutions of constrain-based and score-based approaches are equivalent [2].

A *scoring* function is the key element that makes a search method possible. It provides a measure of quality for a structure to describe datasets sampled from a hypothetical distribution (assumed to be the same used to sample the observed dataset). There are different aproaches for deriving such metrics, but most of them focus on rewarding the structure by the log-likelihood of G according to D and penalizing the structure complexity.

Given a Bayesian Network with structure G and conditional probabilities Θ, and a data set D, we compute its data loglikehood $L_{G,D}$ as:

$$L_{G,D}(\Theta) = \log p(D|G, \theta) \quad (2)$$

$$= \sum_{i=1}^{n} \sum_{x_{Pa_i}} \sum_{x_i} n(x_i, x_{Pa_i}) \log \theta_i(x_i, x_{Pa_i}), \quad (3)$$

where $n(x_i, x_{Pa_i})$ indicates the number of instances (i.e., rows) of D where X_i takes values x_i and each $X_j \in Pa_i$ takes its corresponding value in x_{Pa_i}, and the inner sums are resp. over the values of X_i and Pa_i.

Two commonly used score functions are the BIC score and the BD score and its variants. The BIC score [17] was first proposed as a heuristic for avoinding overfitting introduced by maximizing likelihood and later provided a formal justification as an asymptotic approximation for the marginal posterior distribution of a graph. It is defined as:

$$s_{\text{BIC}}(G) = \max_{\theta} L_{G,D}(\Theta) - \frac{\log N}{2} \cdot |G|, \quad (4)$$

where N is the size of D (i.e., number of rows) and

$$|G| = \sum_{i=1}^{n} (|X_i| - 1) \prod_{X_j \in Pa(X_i)} |X_j| \quad (5)$$

is the number of free parameters ($|X_i|$ denotes the number of values X_i can assume). It is well known that $\arg\max_\theta L_{G,D}(\theta) = n(x_i, x_{Pa_i})/n(x_{Pa_i})$ (these are known as the maximum likelihood estimates of the conditional probability values).

The BD score [3] evaluates a structure by its marginal posterior probability $p(G|D)$ (up to a constant):

$$s_{\text{BD}}(G) = \log p(G) \int_{\theta} p(D|G, \theta) p(\theta|G). \quad (6)$$

The BDe score is a variant of the above metric which assigns a prior distribution over the network parameters that satisfy data likelihood equivalence: two structure with the same data likelihood are assigned the same (BDe) score. The BDeu score imposes an additional assumption that conditional probability distributions are sampled with uniform probability [2].

It is important to note that the BDeu score evaluates the *marginal* likelihood, as opposed to the maximum likelihood. By integrating the parameters out of the equation it measures the expected likelihood averaged over different possible choices of θ [1]. In this way, the BDe score helps avoiding overfitting even more by being less optmistic regarding a particular choice of parameters that maximizes the likelihood only on testing data.

Both these score functions satisfy the decomposability property that states that they can be written as a sum of local score functions $s_i(Pa_i)$. Furthemore, any of these local scores can be computed in time polynomial in the dataset (but exponential in the cardinality of Pa_i). Given the independence relations encoded in a Bayesian Network, the score of a complete network structure can be computed as a sum of scores of each variable given only its parents in the graph. This way, the goal of a search method is to find a structure G^* such that:

$$G^* = \arg\max_{G \in G_{n,k}} \sum_{i \in V} s_i(Pa_i), \tag{7}$$

where $G_{n,k}$ is the set of all possible DAGs with n nodes and treewidth at most k. This can be seen as a classic optmization problem and be solved using known search techniques such as Genetic Algorithms [18] or Hill Climbing [19]. Less obviously, the problem above can be cast as a mixed-integer linear programming optimization [9], [13], [20].

A linear program is a mathematical description of a constrained optimization problem involving linear inequalities over continuous variables and a linear objective. An Integer Linear Program (ILP) is a special case where the variables are restricted to take only integer values. When there is a mix of continuous and integer variables the problem is called a Mixed-Integer Linear Program (MILP). Casting the structure learning problem as a MILP problem allows us to benefit from highly optimized commercial solvers of mathematical programming, and it has been shown to be very effective. In fact, one of the most popular methods for learning Bayesian Networks with no constraint on treewidth is based on a MILP formulation [21], [22].

III. Mixed Integer Linear Program Formulations

In this section we present the MILP formulation for learning Bayesian Network structures proposed in [9]. We assume that the scores for all variables of the network are pre computed and can be retrieved in constant time (a common assumption). We choose using the MILP approach because (i) it is an anytime method, (ii) it is easy to relax the constraing on treewidth, (iii) it allows for other constraints be easily inserted, and (iv) it allows the use of commercial packages, which are readily available and easy-to-use.

Figure 6 gives the integer program formulation. Our constraints are divided in two groups: A group that enforces bounding treewidths (Constrs. 8b to 8d, 8i and 8k) and a group related to learning the network structure (Constrs. 8e to 8l).

A. Bounding treewidth

This formulation aims at encoding all possible elimination orders of a given graph $G = (V, E)$. If a solution exists, a chordalization of the graph with treewidth at most w can be obtained from the integer program.

Variable z_i, which takes real values (Constr. 8i), partially defines the elimination order in the formulation (partially because the formulation allows two nodes to have the same value of z, indicating that any order results in the same chordal graph). A variable i is eliminated before j if $z_i < z_j$.

Maximize:

$$\sum_{it} Pa_{it}.s_i(F_{it}) \tag{8a}$$

Subject to:

$$\sum_{j \in N} y_{ij} \leq w \qquad \forall i \in N \tag{8b}$$
$$(n+1).y_{ij} \leq n + z_j - z_i \qquad \forall i, j \in N \tag{8c}$$
$$y_{ij} + y_{ik} - (y_{jk} + y_{kj}) \leq 1 \qquad \forall i, j, k \in N \tag{8d}$$
$$\sum_{t} Pa_{it} = 1 \qquad \forall i \in N \tag{8e}$$
$$(n+1)Pa_{it} \leq n + v_j - v_i \qquad \forall i \in N, \forall t, \forall j \in F_{it} \tag{8f}$$
$$Pa_{it} \leq y_{ij} + y_{ji} \qquad \forall i \in N, \forall t, \forall j \in F_{it} \tag{8g}$$
$$Pa_{it} \leq y_{jk} + y_{kj} \qquad \forall i \in N, \forall t, \forall j, k \in F_{it} \tag{8h}$$
$$z_i \in [0, n] \qquad \forall i \in N \tag{8i}$$
$$v_i \in [0, n] \qquad \forall i \in N \tag{8j}$$
$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{8k}$$
$$Pa_{it} \in \{0, n\} \qquad \forall i \in N, \forall t \tag{8l}$$

Fig. 6: MILP formulation for learning Bayesian Networks of bounded treewidth

Variables y_{ij} are binary valued (Constr. 8k), and ensure that a node i is eliminated before j ($z_i < z_j$) and an edge existis among them in the resulting chordal graph.

Constraint 8b bound the treewidth to be at most w by bounding the number of high ordered neighbors of every variable. By constrint 8c y_{ij} can only have value 1 if j appears after i in the elimination order. Constraint 8d guarantes the perfect elimination order of z because if variable j and k are higher ordered neighbors of i, then they are also neighbors.

B. Structure learning

Having a chordal graph $G' = (V, E')$, its perfect elimination order and a set of binary valued variables y_{ij} such that it is 1 only if E' contains ij and i is eliminated before j, the rest of our formulation specifies all DAGs over N for which the moral graph is a subgraph of G'.

The set F_i in constraints 8f to 8h is the set of all possible parents sets for that variable. This can be just the subset of $V \setminus \{i\}$, but normally scoring functions can limit this subset or it can even be specified by the user. An element F_{it} of this subset is a set of variables denoting one of those possible parents combinations. Each $\forall t$ in each constraint is for $1, 2, ..., |F_i|$.

The variable v_i, takes real values in $[0, n]$ (Constr. 8j) and specifies a topological order for the variables. If $v_i > v_j$ then j is not an ancestor of i. The $n$ary (constr. 8l) variable Pa_{it} denotes if the tth parent set of F_i were chosen for variable i.

Only one possible parent set can be chosen for each variable, as enforced by constraint 8e and those choices must be acyclic (Constr. 8f). Constraints 8g and 8h ensure that the edges of the found DAG have corresponding edges in G'.

Dataset	Variables	Sample Size
Mushroom	22	8124
WDBC	31	569
Audio	62	200

TABLE I: Datasets

C. Combining formulations

Having the set of variables $y_{ij}, z_i, v_i, Pa_{it}$ with $i, j \in V$, $t = 1, 2, ..., |Pa_i|$ and all satisfying constraints 8a to 8l. A Directed Graph $G = (V, E)$ where $E = \{i \leftarrow j : i, j \in V,\ \exists t\ s.t.\ Pa_{it} \text{ and } j \in F_{it}\}$ is acyclic, consistent with parents Pa_i and has treewidth at most w.

This formulation produces an optimal structure. However, both taks are difficult taks per se and their combination requires significant time and memory resources. Fortunately most MILP optmizers allow runs to be ended prematurely with a valid (perhaps not optimal) solution. In this way, this formulation can also be used to find approximate solution for learning Bayesian Networks. Some solvers even provide an outer error bound, showing the distance between the found solution for the maximum score. This is the approach used in the experiments reported in the next section.

IV. Experiments

In this section we present our methodology and results.

A. Methods

We empirically analyzed the performance of our networks by comparing their likelihood to the data. We used our MILP formulation on a training portion of each dataset to learn the structure and then we learned the network's parameters and calculated their log-likelihood to a test portion of each dataset. We have also tried using the alternative MILP formulation of Parviainen et al. [13], available through the authors' implementation, but we found it to be too slow.

We scored the networks using the BDeu function. We limited the maximum parents of each variable to 4 and used equivalent sample sizes (ESS) of 1.0 and 0.5. The two different values of ESS are used to isolate possible performance differences introduced by the score (and not by the constraint on treewidth). For each variation in each dataset, we learned networks of bounded treewidth, starting with treewidth 3, 4, 5 and then increments of 5 (or 10) until (and including) the dataset's maximum number of variables (which is equivalent as not imposing a bound to the networks's treewidth). For each iteration of treewdith, we ran a 5-fold cross validation to obtain a more realiable estimate of the expected test likelihood.

As mentioned before, our chosen datasets were selected from the UCI Repository[1]. We have selected datasets of different dimensions used in previous works on learning bounded treewidth Bayesian Networks [9], [11], but since our preliminary tests showed that datasets with few variables (say less than 15 variables) usually have "optimal" structures of very small treewidth, such datasets were discarded from this analysis. The datasets with 100 or more variables were also discarded for being computationally too costly to the MILP approach used. We were then left with 3 datasets of different dimensionality. These datasets are summarized in Table I. The experiments were performed by programs wrtiten in Python V.2.7. We used the CPLEX's Python API to interface with CPLEX version 12.6.1 for the MILP problems.

[1] http://archive.ics.uci.edu/ml/

Parents Limit	3	4	
ESS	1	0.5	1
	Avg. GAP	Avg. GAP	Avg. GAP
tw 03	32.42%	62.98%	62.90%
tw 04	26.69%	49.54%	51.38%
tw 05	23.92%	44.06%	45.72%
tw 10	20.27%	32.57%	33.09%
tw 20	20.51%	33.61%	32.26%
tw unlm.	20.10%	32.90%	31.88%

TABLE II: Remaining GAP values for the optimal solution for all MILP executions of the Dataset Mushroom

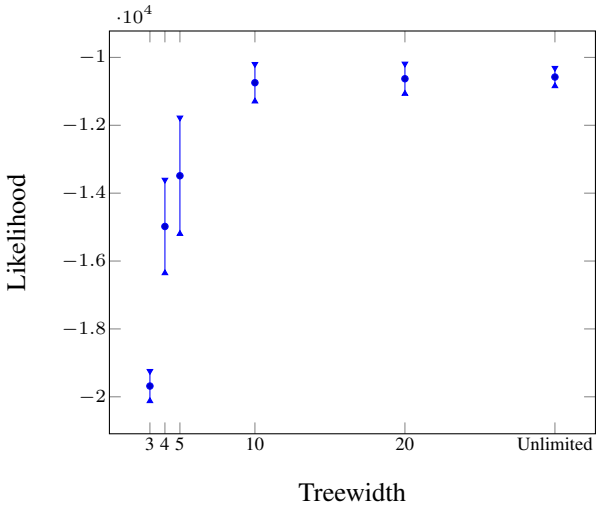


Fig. 7: Likelihood for the networks learned from dataset Mushroom, bounding the parent set to a maximum of 4 and using a ESS score of 1

We called each unique learning experiment a task. For example, the learning experiment of the first fold of the dataset Mushroom with maximum number of parents limited to 4 and the BDeu score's ESS parameter of 1.0 is one task. We ran our experiments in a HPC cluster with nodes of 20 Intel®Xeon®CPU E7-2870 of 2.40GHz and 512GB of available memory. Up to 6 tasks were allowed to run at the same time on each node and each CPLEX optimizer assigned to each task were given three cores to work in parallel. We allowed the optimizer to run up to two hours (CPU time) and we collected the found solution and the error gap of each task.

B. Results

We used the LibTW utility from [23] to calculate the treewidth of the unbounded networks so we can see the underlying treewidth of the "true" network of the training data. Even though learning the true network of the training data is not our main objective, we use this information so we can see the upper bound of the network's treewidth on that dataset.

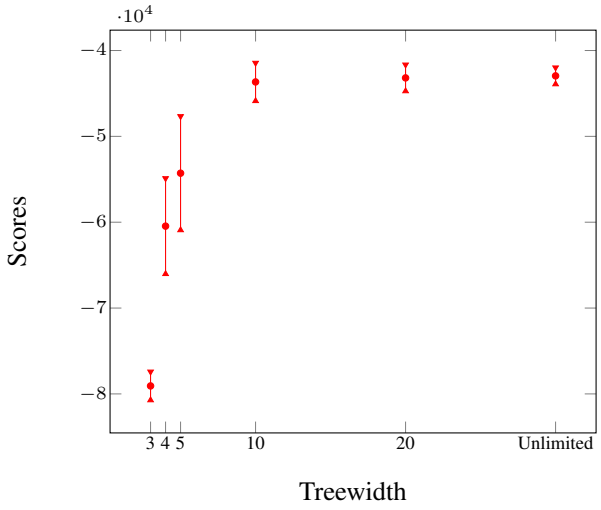


Fig. 8: Scores for the networks learned from dataset Mushroom, bounding the parent set to a maximum of 4 and using a ess score of 1

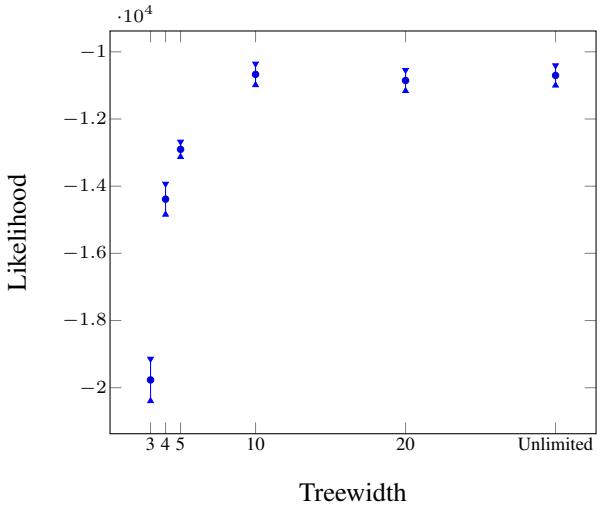


Fig. 9: Likelihood for the networks learned from dataset Mushroom, bounding the parent set to a maximum of 4 and using a ESS score of 0.5

The treewidth of the unbounded network for the dataset Mushroom stayed between 6 and 9 (we give an interval since different folds could yield different treewidths). In fact, our results (Figures 7 and 9) shows that the network‘s performance improves as the treewidth rises and stays at its best between treewidths 5 and 10. Allowing higher treewidths has no impact in the likelihood. By looking at the error gaps in Table II we can see that decreasing the bound on treewidth had a significant impact on the linear program’s performance. If we look at graphs on Figures 8 and 10 we can see that the score of the network follows the same behavior and because of that we can say that the poor performance of the network bounded on treewidth 3 is due to poor ability to learning a good network rather than poor generalization. Varying the ESS parameter on the score function had little differences in the learned networks.

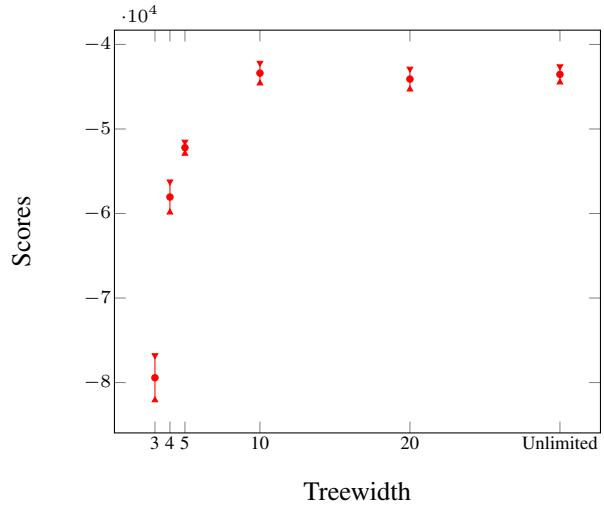


Fig. 10: Scores for the networks learned from dataset Mushroom, bounding the parent set to a maximum of 4 and using a ESS score of 0.5

Parents Limit	3	4	
ESS	1	0.5	1
	Avg. GAP	Avg. GAP	Avg. GAP
tw 03	25.11%	21.25%	27.32%
tw 04	22.71%	20.25%	25.46%
tw 05	23.09%	19.90%	22.82%
tw 10	19.90%	18.94%	20.54%
tw 15	18.98%	17.50%	19.06%
tw 20	18.48%	16.80%	19.10%
tw 25	18.42%	16.58%	18.11%
tw 30	18.26%	16.65%	18.27%
tw unlm.	18.36%	16.54%	18.09%

TABLE III: Remaining GAP values for the optimal solution for all MILP executions of the Dataset WDBC

The overall treewidth of the (unbounded) WDBC dataset stayed between 7 and 9. Again, in Figures 11 and 13 we can see that bounding the treewidth even further impacted the network‘s performance, altough it is hard to say if this is due to poor performance of the network or high complexity in the learning task (see the gap values in Table III). The WDBC dataset showed the same assintotic behavior of Mushroom dataset, but with higher variance, and also, the likelihood of the network with an ESS parameter of 0.5 (Figure 13) showed similar result but a much higher variance.

We couldn‘t find the treewidth of unbounded networks for the audio dataset. As we can see from Figure 15 the audio dataset didn‘t followed the expected assintotic behavior and our learned networks had a significant higher likelihood with low treewidths, even better than the unbounded performance. This shows that, when the learning algorithm can find a good solution, constraining the treewidth can lead to a better generalization. For the same reason, when the learning algorithm can‘t find good solutions (i.e. treewidths 10 to 50 in Figure 15), the network have a poor behavior no matter the treewidth.

We tested the Audio training data with an algorithm that learns network structure with no bound on the treewidth [24]. Since it doen‘t have this restrinction it was expected that the found solution would have a better performance. Indeed,

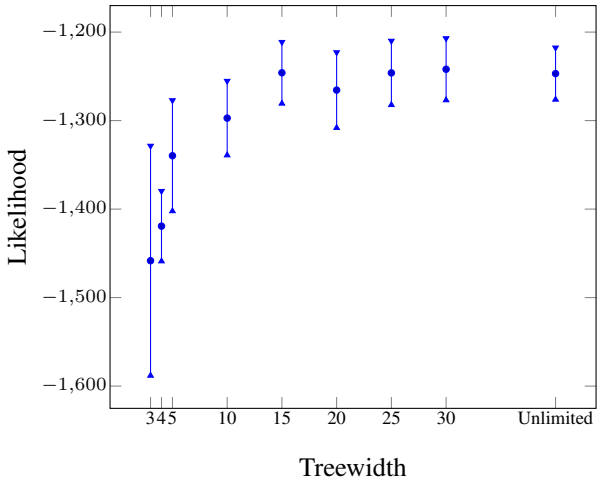


Fig. 11: Likelihood for the networks learned from dataset WDBC, bounding the parent set to a maximum of 4 and using a ESS score of 1

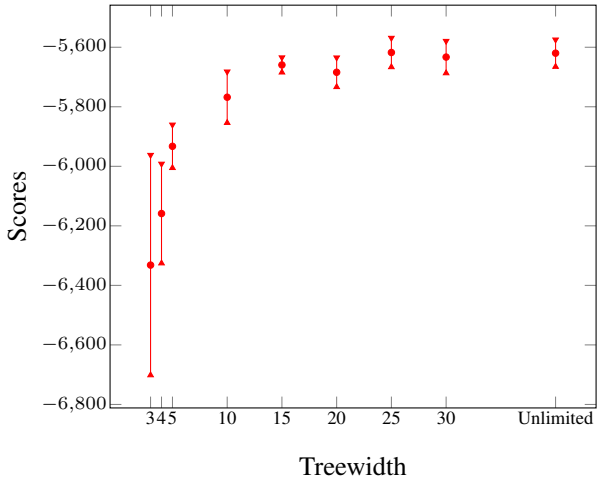


Fig. 12: Scores for the networks learned from dataset WDBC, bounding the parent set to a maximum of 4 and using a ESS score of 1

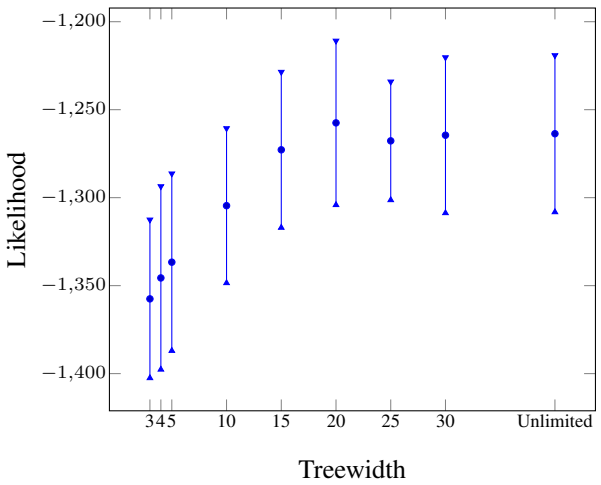


Fig. 13: Likelihood for the networks learned from dataset WDBC, bounding the parent set to a maximum of 4 and using a ESS score of 0.5

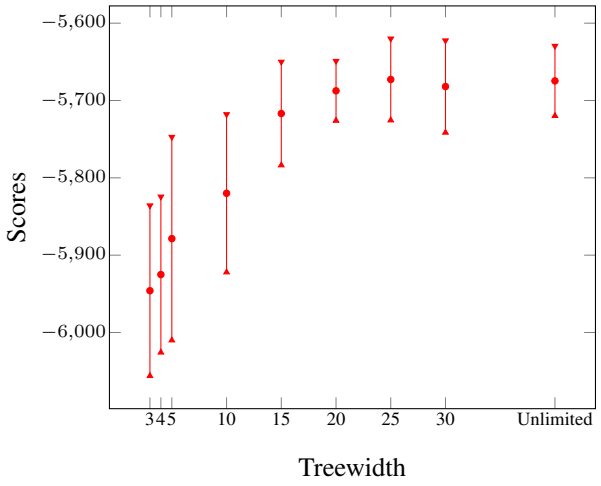


Fig. 14: Scores for the networks learned from dataset WDBC, bounding the parent set to a maximum of 4 and using a ESS score of 0.5

we found likelihood scores of up to 30% better than our learning algorithm but unfortunately, we can't achieve this same performance in leaning structures of bounded treewidth.

V. Conclusion

Overall our results showed that by considering only the likelihood of the learned network to the test data, limiting the treewidth have not outright improved the network's performance as expected. It also showed that the relevance of higher limits on treewidth only starts to matter with high dimension datasets, the same datasets for which the current methods of learning networks of bounded treewidth behave very poorly. Still, more experiments are necessary to distinguish the effect of bounded treewidht and the effect of poor performance of learning algorithms (i.e. experiments that allow a lower the gap on the found solutions). We also suggest a further evaluation of performance by measuring the posterior inference capabilities of the learned network, as well as its bahavior in multilabel classification problems.

Acknowledgment

The first author of this paper was supported by CNPq, grant 165873/2014-0. The third author is partially supported by CNPq. Experiments were made using resources of the LCCA — Laboratory of Advanced Scientific Computation of the University of São Paulo.

References

[1] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge: MIT Press, 2009.

[2] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning {B}ayesian networks: The combination of knowledge and statistical data," *Mach. Learning*, vol. 20, no. 3, pp. 197–243, 1995.

Parents Limit	3	4
ESS	1	1
	Avg. GAP	Avg. GAP
tw 03	13.37%	19.35%
tw 04	14.44%	16.77%
tw 05	14.40%	20.47%
tw 10	28.40%	30.79%
tw 15	28.40%	30.72%
tw 20	28.40%	30.79%
tw 25	28.40%	30.79%
tw 30	25.42%	30.79%
tw 35	25.41%	30.79%
tw 40	22.30%	30.79%
tw 45	20.83%	30.79%
tw 50	17.96%	30.55%
tw 55	14.38%	23.64%
tw 60	12.02%	20.15%
tw unlm.	20.92%	26.92%

TABLE IV: Remaining GAP values for the optimal solution for all MILP executions of the Dataset Audio

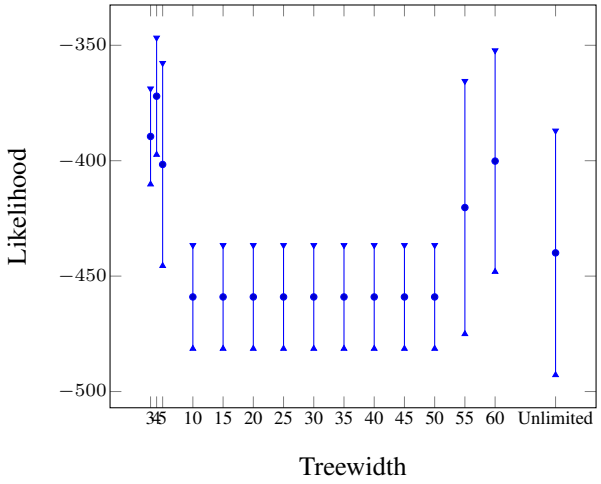


Fig. 15: Likelihood for the networks learned from dataset Audio, bounding the parent set to a maximum of 4 and using a ESS score of 1

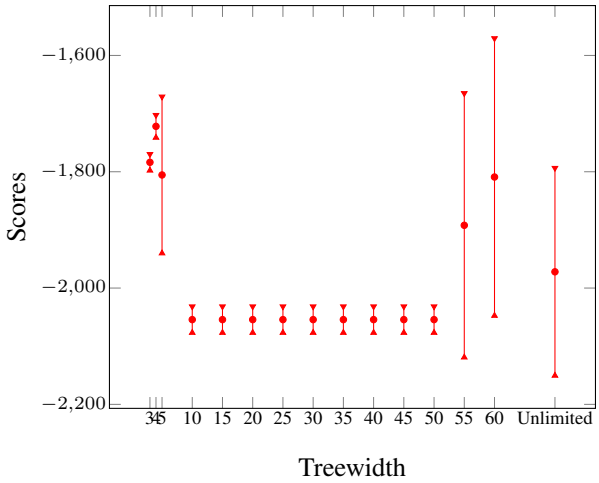


Fig. 16: Scores for the networks learned from dataset Audio, bounding the parent set to a maximum of 4 and using a ESS score of 1

[3] G. F. G. F. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, no. 4, pp. 309–347, Oct. 1992.

[4] A. Antonucci, G. Corani, D. D. Mauá, and S. Gabaglio, "An Ensemble of {B}ayesian Networks for Multilabel Classification," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence ({IJCAI})*, 2013, pp. 1220–1225.

[5] V. Chandrasekaran, N. Srebro, and P. Harsha, "Complexity of Inference in Graphical Models," in *Proc. 24th Conf. on Uncertainty in {AI}*, 2008, pp. 70–78.

[6] 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 *Proc. 19th European Conf. on AI*, 2010, pp. 237–242.

[7] J. Kwisthout, "Treewidth and the Computational Complexity of MAP Approximations," *Probabilistic Graphical Models*, vol. 8754, pp. 271–285, 2014.

[8] R. Gens and D. Pedro, "Learning the Structure of Sum-Product Networks," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 873–880. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/papers/icml2013_gens13

[9] S. Nie, D. D. Maua, C. P. de Campos, Q. Ji, D. D. Mauá, C. P. de Campos, and Q. Ji, "Advances in Learning Bayesian Networks of Bounded Treewidth," in *Advances in Neural Information Processing Systems 27*, Jun. 2014, p. 23.

[10] J. Berg, M. Järvisalo, and B. Malone, "Learning Optimal Bounded Treewidth {B}ayesian Networks via Maximum Satisfiability," in *Proc. 17th Int. Conf. on {AI} and Stat.*, 2014, pp. 86–95.

[11] G. Elidan and S. Gould, "Learning Bounded Treewidth Bayesian Networks," *J. of Mach. Learning Res.*, vol. 9, pp. 2699–2731, 2008.

[12] J. H. Korhonen and P. Parviainen, "Exact Learning of Bounded Tree-width {B}ayesian Networks," in *Proc. 16th Int. Conf. on {AI} and Stat.*, 2013, pp. 370–378.

[13] P. Parviainen, H. S. Farahani, and J. Lagergren, "Learning Bounded Tree-width Bayesian Networks using Integer Linear Programming," in *Proc. 17th Int. Conf. on {AI} and Stat.*, 2014, pp. 751–759.

[14] S. Arnborg, D. Corneil, and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. on Matrix Analysis and Applications*, vol. 8, no. 2, pp. 277–284, 1987.

[15] M. Teyssier and D. Koller, "Ordering-based search: A simple and effective algorithm for learning Bayesian networks," in *Proc. 21st Conf. on Uncertainty in {AI}*, 2005, pp. 584–590.

[16] N. Robertson and P. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," *Journal of algorithms*, vol. 7, no. 3, pp. 309–322, 1986.

[17] G. Schwarz, "Estimating the dimension of a model," *Annals of Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

[18] P. Larranaga and C. Kuijpers, "Learning Bayesian network structures by searching for the best ordering with genetic algorithms," *Systems, Man and Cybernetics*, vol. 26, no. 4, pp. 487–493, 1996.

[19] I. Tsamardinos, L. Brown, and C. Aliferis, "The max-min hill-climbing Bayesian network structure learning algorithm," *Machine learning*, vol. 65, no. 1, pp. 31–78, 2006.

[20] A. Grigoriev, E. Hans, and U. Natalya, "Integer linear programming formulations for treewidth," Maastricht University, Maastricht Research School of Economics of Technology and Organization (METEOR), Tech. Rep., 2011.

[21] M. Bartlett and J. Cussens, "Integer Linear Programming for the Bayesian network structure learning problem," in *Artificial Intelligence*, Mar. 2015, p. 14.

[22] J. Cussens, "Bayesian network learning with cutting planes," in *Proc. 27th Conf. on Uncertainty in AI*, 2011, pp. 153–160.

[23] T. van Dijk, J. van den Heuvel, and W. Slob, "Computing treewidth with LibTW," University of Utrecht, Tech. Rep., 2006.

[24] M. Barlett and J. Cussens, "Advances in Bayesian Network Learning using Integer Programming," in *Proc. 29th Conf. on Uncertainty in AI*, 2013, pp. 182–191.