Learning Large Bayesian Networks with Expert Constraints

Abstract

We propose a new score-based algorithm for learning the structure of a Bayesian Network (BN). It is the first algorithm that simultaneously supports the requirements of (i) learning a BN of bounded treewidth (ii) satisfying expert constraints including positive and negative ancestry properties between nodes, and (iii) scaling up to BNs with several thousand nodes.

The algorithm operates in two phases. In Phase 1, we utilize a modified version of the k-greedy algorithm by Scanagatta et al. (NeurIPS'16), modified to generate an initial DAG that supports a portion of the given constraints. In Phase 2, we follow the BN-SLIM framework, introduced by Peruvemba Ramaswamy and Szeider (AAAI'21). We improve the initial DAG by repeatedly running a MaxSAT solver on selected local parts. The MaxSAT encoding entails local versions of the expert constraints as hard constraints.

We evaluate a prototype implementation of our algorithm on several standard benchmark sets. The encouraging results demonstrate the power and flexibility of the BN-SLIM framework. It boosts the score while increasing the number of satisfied expert constraints.

1 INTRODUCTION

Bayesian network structure learning is the computationally expensive problem of discovering a Bayesian network (BN) that optimally represents a given training data set [Chickering, 1996]. In addition to fitting the data, often measured in terms of a score function, several other requirements have been taken into account for the BN structure learning.

A fundamental requirement considered by an extensive volume of research is to learn BNs that fit the data and have *bounded treewidth* [Benjumeda et al., 2019, Berg et al., 2014, Elidan and Gould, 2009, Nie et al., 2015, Scanagatta et al., 2016, 2018, Korhonen and Parviainen, 2013, Parviainen et al., 2014]. Bounded treewidth BNs admit tractable probabilistic inference [Kwisthout et al., 2010].

Another fundamental requirement, receiving a growing amount of attention, is to learn BNs that fit the data and satisfy additional *expert constraints* [Chen et al., 2016, Kennett et al., 2001, Li and van Beek, 2018, Corander et al., 2013]. Such constraints can assert, for instance, direct or indirect causation between random variables in terms of whether or not one variable is a parent or an ancestor of the other in the DAG of the learned BN. See Table 1 for a list of expert constraints considered in the literature.

Table 1: Various expert or side constraints considered in the literature.

Arc constraints (direct causation)	
$u \to v$	the DAG contains the arc (u, v)
$u \not\to v$	the DAG does not contain the arc (u, v)
$u \leftrightarrow v$	the DAG contains either the arc (u, v) or the arc (v, u)
Ancestry constraints (indirect causation)	
$u \rightsquigarrow v$	the DAG contains a path from u to v
$u \not\rightsquigarrow v$	the DAG does not contain a path from u to v
$u \leftrightsquigarrow v$	the DAG contains either a path from u to v or one in the other direction

In addition to bounded treewidth and expert constraint requirements, one needs to address the *scalability* of methods for BN structure learning. For instance, learning a BN of bounded treewidth that optimally fits the data is NP-hard [Korhonen and Parviainen, 2013]. The consideration of expert constraints provides an additional source of complexity.

In this paper, we propose Con-BN-SLIM, the first method for BN structure learning that addresses all three require-

Submitted to the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022). **To be used for reviewing only**.

ments simultaneously: bounded treewidth, expert constraints, and scalability. Table 2 shows how our new method compares to other BN structure learning methods from the literature.

Con-BN-SLIM consists of two phases.

In Phase 1, a heuristic greedily computes a candidate BN from data, thereby trying to satisfy as many expert constraints as possible. The heuristic is a version of k-greedy by Scanagatta et al. [2018] that we modified to consider expert constraints. This method scales very well. However, the consideration of expert constraints significantly deteriorates the algorithm's capability of fitting the BN to the data. This even prevails when we consider the expert constraints as soft constraints, which allows the algorithm to violate some constraints.

We therefore add a Phase 2, that takes the candidate BN from the first phase and repeatedly tries to improve the score by optimizing local parts of the BN. The second phase is an extension of the BN-SLIM approach by Peruvemba Ramaswamy and Szeider [2021a]. BN-SLIM utilizes a MaxSAT solver to locally improve the BN. Crucial for our extension is to be able to express suitable local versions of the desired expert constraints in terms of hard constraints for the MaxSAT solver. This way, the solver may improve the fitting of the BN while maintaining the satisfaction of all the expert constraints satisfied by the first phase solution.

We evaluated a prototype implementation of Con-BN-SLIM on all discrete sample data from the Bnlearn repository, sampling expert constraints from the ground truth networks. After the first phase of running the modified heuristics for about 30 minutes, the rate of improvements deteriorates. Phase 2 begins, and Con-BN-SLIM takes over the candidate network and shows a remarkably high improvement rate. The final network shows a significantly higher score than the network produced by Phase 1, and this displays favorably in the ΔBIC metric.

The empirical findings on our prototype implementation are highly encouraging, providing the ground for several avenues of further investigation.

2 PRELIMINARIES

In this section, we provide a brief overview of the required background. We closely follow the general notation and methodology of Peruvemba Ramaswamy and Szeider [2021a] throughout this section.

2.1 STRUCTURE LEARNING

We consider the problem of learning the structure (i.e., the DAG) of a BN from complete data set of N instances $D_1, \ldots, D_N$ over a set of n categorical random variables $X_1, \ldots, X_n$. The goal is to find a DAG $D = (V, E)$ where V is the set of nodes (one for each random variable) and E is the set of arcs (directed edges). The value of a *score function* determines how well a DAG D fits the data; the DAG D, together with local parameters, forms the BN Koller and Friedman [2009].

We assume that the score is *decomposable*, i.e., being constituted by the sum of the individual random variables' scores. Hence we can assume that the score is given in terms of a *score function* f that assigns each node $v \in V$ and each subset $P \subseteq V \setminus \{v\}$ a real number $f_P(v)$, the *score* of P for v. The score of the entire DAG $D = (V, E)$ is then $f(D) := \sum_{v \in V} f(v, P_D(v))$ where $P_D(v) = \{\, u \in V : (u, v) \in E \,\}$ denotes the *parent set* of v in D. This setting accommodates several popular scores like AIC, BDeu, and BIC Akaike [1974], Heckerman et al. [1995], Schwarz [1978]. If P and P' are two potential parent sets of a random variable v such that $P \subsetneq P'$ and $f(v, P') \leq f(v, P)$, then we can safely disregard the potential parent set P' of v. Consequently, we can disregard all nonempty potential parent sets of v with a score $\leq f(v, \emptyset)$. Such a restricted score function is a *score function cache*.

2.2 TREEWIDTH

A *tree decomposition* $\mathcal{T}$ of a graph G is a pair (T, χ), where T is a tree and χ is a function that assigns each tree node t a set $\chi(t)$ of vertices of G such that the following conditions hold:

T1 For every edge e of G there is a tree node t such that $e \subseteq \chi(t)$.

T2 For every vertex v of G, the set of tree nodes t with $v \in \chi(t)$ induces a non-empty subtree of T.

The sets $\chi(t)$ are called *bags* of the decomposition $\mathcal{T}$, and $\chi(t)$ is the bag associated with the tree node t. The *width* of a tree decomposition (T, χ) is the size of a largest bag minus 1. The *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of G.

The *treewidth-bounded BN structure learning problem* takes as input a set V of nodes, a decomposable score function f on V, and an integer W, and it asks to compute a DAG $D = (V, E)$ of treewidth $\leq W$, such that $f(D)$ is maximal.

2.3 EXPERT CONSTRAINTS

In our work, we consider only arc and ancestry constraints. The requirements for satisfaction of the constraints is described in Table 1. We use the term *constraint set* to refer to a set of such constraints and a DAG D is said to satisfy a constraint set if it satisfies all constituent constraints. We refer to $\rightarrow$ and $\rightsquigarrow$ as *positive* constraints and $\not\rightarrow$ and $\not\rightsquigarrow$ as *negative* constraints. Note that, $u \not\rightsquigarrow v$ is denoted as $v > u$

Table 2: Feature Comparison Table. † CaMML allows weighted constraints with weight of 1 signifying hard constraints. ‡ Negative constraints are treated as hard constraints while positive constraints can be violated.

	scalability (# RVs)	bounded treewidth	supported constraints	score optimization
EC Tree [Chen et al., 2016]	≤ 20	no	$\{\rightsquigarrow, \not\rightsquigarrow\}$	exact
MINOBSx [Li and van Beek, 2018]	≤ 50	no	$\{\not\rightarrow, \not\rightsquigarrow, \rightarrow, \leftrightarrow, \rightsquigarrow\}$	approx.
CaMML [Kennett et al., 2001]	unknown	no	$\{\not\rightsquigarrow, \rightarrow, \leftrightarrow, \rightsquigarrow\}^{\dagger}$	exact
k-greedy [Scanagatta et al., 2018]	≤ 10000	yes	$\varnothing$	approx.
BN-SLIM [PR and Szeider, 2021a]	≤ 10000	yes	$\varnothing$	approx.
Con-k-greedy (this paper)	≤ 10000	yes	$\{\not\rightarrow, \not\rightsquigarrow, \rightarrow, \leftrightarrow, \rightsquigarrow\}^{\ddagger}$	approx.
Con-BN-SLIM (this paper)	≤ 10000	yes	$\{\not\rightarrow, \not\rightsquigarrow, \rightarrow, \leftrightarrow, \rightsquigarrow\}^{\ddagger}$	approx.

by Li and van Beek [2018]. Also note that, some other variants of constraints like $\not\leftrightsquigarrow$ can be expressed as boolean combinations of the elementary constraints from Table 1.

Given a DAG over a set V of vertices, a path P is a sequence $v_1, \ldots, v_\ell$ of vertices such that there exists an arc from v_i to v_{i+1} for all $1 \leq i < \ell$. A path is said to be simple if all involved v_i are distinct. In this paper, we refer to simple and directed paths simply as *paths*. The path P *avoids* a set $S \subseteq V$ if $v_i \notin S$ for all $1 \leq i < \ell$.

Finally, we also use the concept of partial orders in our modification of k-greedy (Section 3). A partial order is a set of pairwise ordering requirements $u \triangleright v$. A linear order $u_1, \ldots, u_n$ is said to *obey* a partial order if, for every $u_i \triangleright u_j$ in the partial order, $i < j$.

3 K-GREEDY WITH CONSTRAINTS

In this section we describe the modifications made to k-greedy in order to obtain a heuristic algorithm to solve the Constrained BN structure learning problem. We would like to point out that we chose to modify k-greedy as a proof of concept because of its simplicity. However, theoretically similar modifications are also possible for the more aggressive k-MAX heuristic [Scanagatta et al., 2018].

3.1 OVERVIEW OF K-GREEDY

First, we give a brief overview of basic k-greedy heuristic [Scanagatta et al., 2016]. The algorithm takes as input a set X of RVs and a score function cache and returns a DAG D along with a corresponding (rooted) tree decomposition T. The algorithm repeatedly performs the following steps:

Step 1. Randomly sample a linear ordering σ over the variables X

Step 2. Construct the root bag of T from the first $k+1$ variables of σ. Also compute a DAG over these variables maximizing the score (either exactly or approximately).

Step 3. Then insert the remaining variables from σ one by one into the DAG, selecting the best parent set for it from the already inserted variables.

After each step, if the newly computed DAG has a higher score than the previous best DAG, it is called an *improvement*.

3.2 MODIFICATION

In order to upgrade this algorithm to work with expert constraints, we modify each of the steps above. In Step 1, instead of randomly sampling an order, we first 'compile' the supplied constraints into a partial order. Meaning, we add a partial order pair $u \triangleright v$ for every positive constraint $u \bowtie v$, i.e., $\bowtie \in \{\rightarrow, \rightsquigarrow\}$. As any topological ordering of any network that satisfies these constaints also obeys this partial order. We then randomly sample linear orderings that obey this partial order. In Step 2, we now search for a best DAG which satisfies the provided constraints either by brute force or using any other local solver. In Step 3, we select the best parent set from among the parent sets that satisfy the given constraints.

3.3 PRACTICAL CONSIDERATIONS

The modified algorithm proposed above only produces solutions which satisfy *all* the given constraints. In other words, it treats all constraints as *hard* constraints. In practice, we noticed this severely limits the number of improvements and in many cases fails to find any networks. We thus, slightly alter Step 3 to only reject choices of parent sets which violate the negative constraints, i.e., $\{\not\rightsquigarrow, \not\rightarrow\}$. As a result, the heuristic provides solutions which satisfy all the negative constraints but not necessarily all the positive and undirected constraints. In other words, all positive and undirected constraints, i.e., $\{\rightarrow, \leftrightarrow, \rightsquigarrow\}$, are treated as *soft* constraints.

Further, in our implementation of Con-k-greedy, we were not able to integrate a constraint-sensitive local solver into k-greedy. This has the implication that negative con-

straints can also be violated, but only if both endpoints lie within the root bag of T.

3.4 EXPERIMENTS

We experimentally evaluated the heuristic proposed above and found the results to be unsatisfactory. We noticed that the rate of improvements diminshes quite quickly and essentially reaches saturation by 30 mins (see Figure 1). However, the output of the heuristic could serve as a starting point for further improvement. The *SAT-based Local Improvement Method* (SLIM) framework introduced by Lodha et al. [2016, 2019] and later used by Fichte et al. [2017], Peruvemba Ramaswamy and Szeider [2020, 2021a,b], Schidler and Szeider [2021] could potentially turbocharge and improve the score of such an intermediate saturated solution. In the next sections, we develop a solution using the SLIM framework for the Constrained BN structure learning problem.

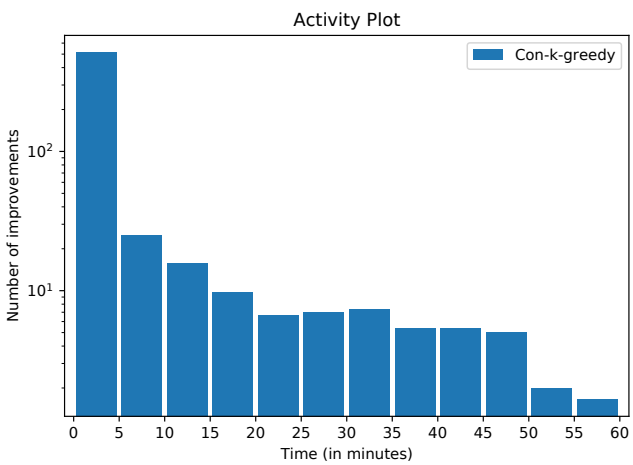

Figure 1: Activity plot showing the rate of improvements of Con-k-greedy against time. Note that the y-axis is in logscale.

4 SLIM WITH CONSTRAINTS

4.1 THEORY

In this section, we lay the theoretical foundation for solving the Constrained BN structure learning problem using the SLIM framework. The SLIM framework has been previously used by Peruvemba Ramaswamy and Szeider [2021a] to solve the BN structure learning problem. We refer to this method as BN-SLIM. We directly extend BN-SLIM to solve the Constrained BN structure learning problem, as a result we reuse the same notation.

The problem input consists of a set V of random variables, a score function f, a treewidth bound W and a set of expert constraints $\mathcal{C}$. We allow $\mathcal{C}$ to contain constraints of type $\{\rightarrow, \not\rightarrow, \leftrightarrow, \rightsquigarrow, \not\rightsquigarrow\}$.

The goal is to compute a DAG $D^\star$ over V with maximum score such that the treewidth of the moralized graph $M(D^\star)$ is bounded by W and $D^\star$ satisfies all the constraints in $\mathcal{C}$. We assume to have an initial heuristic solution D, a corresponding tree decomposition $\mathcal{T} = (T, \chi)$ of width $\leq W$ of the moralized graph $M(D)$ and that D satisfies the constraint set $\mathcal{C}$. Our aim now, is to compute a DAG D^{new} over V with score at least as much as D while still having bounded treewidth and satisfying constraint set $\mathcal{C}$. Applying this process repeatedly we can improve the score of the resultant DAG while still satisfying all the requirements.

We select a subtree $S \subseteq T$ such that the total number of vertices in $V_S := \bigcup_{t \in S} \chi(t)$ is at most some *budget* B. We define D_S^{new} as the DAG induced by D^{new} on V_S, where $E(D_S^{\text{new}}) = \{\, (u, v) \in E(D^{\text{new}}) : \{u, v\} \subseteq V_S \,\}$. and $\mathcal{S}^{\text{new}} = (S^{\text{new}}, \chi^{\text{new}})$ as a tree decomposition of D_S^{new}. For convenience, we use the shorthand E_S^{new} to denote $E(D_S^{\text{new}})$.

We distinguish between different kinds of vertices:

- $v \in V_S$ a *boundary vertex* if there exists a tree node $t \in V(T) \setminus V(S)$ such that $v \in \chi(t)$;
- $v \in V_S$ is an *internal vertex* if v is not a boundary vertex;
- $v \in V \setminus V_S$ is an *external vertex*.

Two boundary vertices v, v' are *adjacent* if both occur together in some bag outside S. In that case we call $\{v, v'\}$ a *virtual edge*. We let E_{virt} be the set of all virtual edges. The *extended moral graph* M_{ext} is obtained from $M(D_S^{\text{new}})$ by adding all virtual edges. If v, v' are two adjacent boundary vertices such that D^{new} contains a directed path from v' to v, where all the vertices on the path, except for v' and v, are external, then (v', v) is a *virtual arc*. $E_{\text{virt}}^{\rightarrow}$ denotes the set of all virtual arcs.

We can now reiterate the conditions from Peruvemba Ramaswamy and Szeider [2021a] needed to state the main theorem.

C1 D_S^{new} is acyclic.

C2 The moral graph $M(D_S^{\text{new}})$ has treewidth $\leq W$.

C3 $\mathcal{S}^{\text{new}}$ is a tree decomposition of the extended moral graph M_{ext}.

C4 For each $v \in V_S$, if $P_{D^{\text{new}}}(v)$ contains external vertices, then there is some $t \in V(T) \setminus V(S)$ such that $P_{D^{\text{new}}}(v) \cup \{v\} \subseteq \chi(t)$.

C5 The digraph $(V_S, E_S^{\text{new}} \cup E_{\text{virt}}^{\rightarrow})$ is acyclic.

Theorem 1 ([Peruvemba Ramaswamy and Szeider, 2021a]). *If all the conditions C1–C5 are satisfied, then D^{new} is*

acyclic, the treewidth of $M(D^{\text{new}})$ is at most W, and the score of D^{new} is at least the score of D.

We now discuss how the different types of constraints can be transformed into their respective local versions along with the correctness for the same. We note that the input constraint set can only consist of elementary arc and ancestry constraints (listed in Table 1), however, the translation into their respective local versions additionally allows disjunctions over elementary constraints. This is because the local versions of the constraints are directly handed off to the MaxSAT solver which is capable of handling such disjunctions.

To discuss the behaviour of the ancestry constraints we use the concept of first-hit descendants and first-hit ancestors. Given a DAG F over vertices W, subset $Y \subsetneq W$ and vertex $r \in W$, a node $s \in Y$ is said to be a first-hit descendant of r in Y if there exists a directed path from r to s avoiding all the other vertices in $Y \setminus \{r, s\}$. We denote by $\text{desc}^r_Y \subseteq Y$, the set of all *first-hit descendants of r in Y*. Similarly, a node $s \in Y$ is said to be a first-hit ancestor of r in Y if there exists a directed path from s to r avoiding all the other vertices in $Y \setminus \{r, s\}$. We denote by $\text{anc}^r_Y \subseteq Y$, the set of all *first-hit ancestors of r in Y*. We denote by $\top$, the always-true trivial constraint that is always satisfied.

Arc constraints ($\rightarrow, \not\rightarrow, \leftrightarrow$) Let c be a constraint $u \bowtie v$, where $\bowtie \in \{\rightarrow, \not\rightarrow, \leftrightarrow\}$. If either of $u, v \notin S$, then the constraint remains satisfied, since the presence or absence of the arc between u, v is not affected by D^{new}. The local version of such a constraint is thus $\top$. Alternatively, if both $u, v \in S$, it suffices to ensure that the constraint c holds in D^{new}_S. The local version of such a constraint is c itself.

Positive ancestry constraints ($\rightsquigarrow$) Consider a constraint of the form $u \rightsquigarrow v$. Since the constraint is satisfied in D, we know that there exists a $u - v$ path in D.

Case 1 There is at least one $u - v$ path avoiding V_S. The constraint remains satisfied independent of D^{new}_S. The local version of such a constraint is $\top$.

Case 2 All $u - v$ paths pass through V_S. It suffices to ensure that there exists at least one path in D^{new}_S from some $d_u \in \text{desc}^u_{V_S}$ to some $a_v \in \text{anc}^v_{V_S}$. The local version of such a constraint is $\bigvee_{d_u, a_v} d_u \rightsquigarrow a_v$.

Negative ancestry constraints ($\not\rightsquigarrow$) Consider a constraint of the form $u \not\rightsquigarrow v$. Since the constraint is satisfied in D, we know that there are no $u - v$ paths in D. Any $u - v$ path passing through V_S must be of the form $u - d_u - a_v - v$, for some $d_u \in \text{desc}^u_{V_S}, a_v \in \text{anc}^v_{V_S}$.

Case 1 $\text{desc}^u_{V_S} = \emptyset$ or $\text{anc}^v_{V_S} = \emptyset$. The constraint remains satisfied independent of D^{new}_S. The local version of such a constraint is $\top$.

Case 2 Both sets are non-empty. It suffices to ensure that there is no path in D^{new}_S from any $d_u \in \text{desc}^u_{V_S}$ to any $a_v \in \text{anc}^v_{V_S}$. The local version of such a constraint is $\bigwedge_{d_u, a_v} d_u \not\rightsquigarrow a_v$.

From this discussion, we can assert the following lemma.

Lemma 1. *If D^{new}_S satisfies the local versions of each of the constraint in $\mathcal{C}$, then D^{new} satisfies the constraint set $\mathcal{C}$.* □

From Theorem 1 and Lemma 1, we obtain the following corollary.

Corollary 1. *If conditions C1–C5 are satisfied and D^{new}_S satisfies the local versions of the constraints in $\mathcal{C}$, then D^{new} is acyclic, the treewidth of $M(D^{\text{new}})$ is at most W, the score of D^{new} is at least that of D and D^{new} satisfies the constraint set C.*

4.2 ENCODING

In this section we describe the MaxSAT encoding to compute D^{new}_S. We build on top of the encoding by Peruvemba Ramaswamy and Szeider [2021a]. Briefly, the basic variables in the encoding are the par^P_v variables, which are true if and only if P is the parent set of v. These variables appear in the encoding as soft clauses weighted by $f(v, P)$. In addition to that, there are several hard clauses involving $\text{arc}_{u,v}$, $\text{acyc}_{u,v}$ and $\text{ord}_{u,v}$ variables, which encode the edges of the moralized graph, the acyclicity of the DAG and the elimination ordering corresponding to the tree decomposition respectively. The soft and hard clauses of the encoding are passed to the MaxSAT solver to optimize for the score of the network. The MaxSAT solver then finds solutions satisfying all the hard clauses while also maximizing the weight of the satisfied soft clauses. Eventually, this encoding finds a network with maximum score that satisfies the conditions C1–C5. For the sake of brevity, we skip repeating the entire encoding and only describe the additions.

Now, having Corollary 1, we describe the addition to the encoding that ensures that D^{new}_S satisfies the local versions of the constraints.

Arc constraints ($\rightarrow, \not\rightarrow, \leftrightarrow$) We filter out the infeasible parent sets based on the arc constraints. More specifically, for the constraint $u \rightarrow v$, we discard all parent sets of v that do not contain u, and conversely, for the constraint $u \not\rightarrow v$, we discard all parent sets that contain u.

Ancestry constraints ($\rightsquigarrow, \not\rightsquigarrow$) We address the ancestry constraints by introducing the following variables to keep track of the paths within the network:

1. $\text{dagarc}_{u,v}$ represents an arc in the DAG from u to v (does not include the moralized and fill-in edges unlike $\text{arc}_{u,v}$),
2. $\text{tarc}_{u,v}$ captures the transitive closure of the $\text{dagarc}_{u,v}$ variables.

3. $\text{path}_{u,v}$ implies the existence of a path in the DAG from u to v,
4. $\text{pathq}_{u,v,z}$ is a helper variable for $\text{path}_{u,v}$ and implies the existence of a path in the DAG from u to v with z as the penultimate vertex,
5. $\text{virtarc}_{u,v}$ represents the short-circuited directed paths through nodes outside the local instance.

We then introduce hard clauses over these variables to capture their semantics and to allow expressing expert constraints. At times, we write the clauses using the friendlier implication notation. However, all of these clauses can be converted into the standard *Conjunctive Normal Form* (CNF) as required by the MaxSAT solver. For this reason, the encoding accepts as input all the elementary constraints as well as disjunctions over elementary constraints.

To disallow simulateneous arcs in opposite directions in the DAG, we add the clauses

$$\neg\text{dagarc}_{u,v} \vee \neg\text{dagarc}_{v,u} \qquad \text{for all } u \neq v \in S.$$

We then add the following clauses to ensure that $\text{dagarc}_{u,v}$ is true if and only if u is in the parent set of v.

$$\text{par}_v^P \Rightarrow \bigwedge_{u \in P} \text{dagarc}_{u,v} \quad \text{for all } v \in S, P \in \mathcal{P}_v, \text{ and}$$

$$\text{dagarc}_{u,v} \Rightarrow \bigvee_{P \in \mathcal{P}_v \text{ s.t. } u \in P} \text{par}_v^P \quad \text{for all } u \neq v \in S.$$

And finally, we propagate the DAG arcs to the arcs of the moralized graph using the clauses

$$\text{dagarc}_{u,v} \Rightarrow \text{arc}_{u,v} \qquad \text{for all } u \neq v \in S.$$

For the $\text{tarc}_{u,v}$ variables, we initialize the transitivity using the $\text{dagarc}_{u,v}$ and $\text{virtarc}_{u,v}$ variables as follows

$$\left.\begin{aligned} \text{dagarc}_{u,v} &\Rightarrow \text{tarc}_{u,v} \\ \text{virtarc}_{u,v} &\Rightarrow \text{tarc}_{u,v} \end{aligned}\right\} \text{ for all } u \neq v \in S,$$

and then encode the transitivity using the following clauses

$$\text{tarc}_{u,v} \wedge \text{tarc}_{v,w} \Rightarrow \text{tarc}_{u,w} \quad \text{for all distinct } u, v, w \in S.$$

To encode the path variables, we first encode the condition that the path can either be a single arc in the DAG, a single external virtual arc or a path through at least one other variable z. For this we add the following clauses for all $u \neq v \in S$,

$$\text{path}_{u,v} \Rightarrow \text{dagarc}_{u,v} \vee \text{virtarc}_{u,v} \vee \bigvee_{z \neq u,v} \text{pathq}_{u,v,z}.$$

Then, we encode the condition for the existence of a path from u to v with z in the penultimate position, by asserting a path from u to z and either a direct arc or a virtual arc from z to v. For this we add the following clauses for all distinct $u, v, z \in S$,

$$\text{pathq}_{u,v,z} \Rightarrow \text{path}_{u,z} \wedge (\text{dagarc}_{z,v} \vee \text{virtarc}_{z,v}).$$

Finally, we encode the constraints using the predicates described so far. For the arc constraints, we use the $\text{dagarc}_{u,v}$ variables as follows

$$\begin{aligned} &\text{for } u \to v, \text{ we use } \text{dagarc}_{u,v}, \\ &\text{for } u \not\to v, \text{ we use } \neg\text{dagarc}_{u,v}, \\ &\text{for } u \leftrightarrow v, \text{ we use } \text{dagarc}_{u,v} \vee \text{dagarc}_{v,u}. \end{aligned}$$

For the ancestry constraints, we use the $\text{path}_{u,v}$ and $\text{tarc}_{u,v}$ variables as follows

$$\begin{aligned} &\text{for } u \rightsquigarrow v, \text{ we use } \text{path}_{u,v}, \\ &\text{for } u \not\rightsquigarrow v, \text{ we use } \neg\text{tarc}_{u,v}, \end{aligned}$$

It is subtle but worth noting nonetheless, that the clause $\neg\text{path}_{u,v}$ does not ensure the absence of a path from u to v in the DAG, i.e., the $\text{path}_{u,v}$ variables can only be used to assert the existence of paths ($\rightsquigarrow$ constraints), not their absence ($\not\rightsquigarrow$ constraints). Which is why we use the $\text{tarc}_{u,v}$ variables to be able to assert the absence of paths.

5 EXPERIMENTS

5.1 SETUP

We tested the two proposed heuristics on 4-core Intel Xeon E5540 2.53 GHz CPU cluster, with each process having access to 8GB RAM. The k-greedy algorithm is available as a part of the BLIP package [Scanagatta, 2015] implemented in Java. We provide the source code for Con-k-greedy as a part of the supplementary material. We implemented Con-BN-SLIM by extending the publicly available BN-SLIM [Peruvemba Ramaswamy and Szeider, 2021c] Python software, which makes heavy use of the NetworkX library [Hagberg et al., 2008], and the UWrMaxSat[1] as the MaxSAT-solver.

We ran the heuristics on score function caches and constraints sets generated from all the discrete networks available as a part of the Bnlearn repository.[2] This repository is commonly used for benchmarking Bayesian Networks [Li and van Beek, 2018, Chen et al., 2016, Scanagatta et al., 2018, 2016]. We split up the networks into three buckets - small, medium, and, large based on the number of random

[1] https://maxsat-evaluations.github.io/2019/descriptions.html

[2] https://www.bnlearn.com/bnrepository/

variables. We then synthesized expert constraints by randomly sampling a fixed number η of constraints of each of the 5 types from the ground truth networks (see Table 3). Note that this repository consists of the networks themselves, not the instances or samples drawn from the BNs. Additionally, we also precomputed the treewidths of all the ground truth networks (ranging between 3 and 15) and used those values as the bounds for all the heuristics.

Table 3: Input Datasets

Bucket	Variables	η
Small	up to 50	$\{5, 10\}$
Medium	50 to 500	$\{10, 25, 50\}$
Large	above 500	$\{25, 50, 75\}$

5.2 METHOD

We now explain the format of the experiments used to compare the proposed heuristics, which is similar to that of Peruvemba Ramaswamy and Szeider [2021a]. We precompute the score function caches using the available functionality from the BLIP package. All the evaluated methods are supplied the same score function caches. We then randomly synthesized different constraints using three random seed values. The score function caches along with a corresponding constraint set are together considered to be one input instance. This results in a total of 183 input instances.

We then ran the original k-greedy algorithm and the Con-k-greedy algorithm on these inputs for 60 minutes. For each input, we ran the heuristics with three different random seed values. For evaluating Con-BN-SLIM, we used the intermediate solution produced by Con-k-greedy at the 30-minute mark as the starting heuristic. After which, we run Con-BN-SLIM for another 30 minutes, thereby fixing the total runtime of each method to 60 minutes. For each input, we ran Con-BN-SLIM with 8 different configurations (random seed, timeout, encoding type). For all the experiments, we record the final score, the final satisfied constraint count and the rate of improvements.

5.3 RESULTS

As a continuation to Figure 1, we first visualize the activity of Con-BN-SLIM in comparison to Con-k-greedy. Note that, Con-BN-SLIM only starts running at the 30-minute mark (after being handed the heuristic solution from Con-k-greedy) and hence does not record any improvements till that point. As is evident from Figure 2, despite the rate of improvements of Con-k-greedy slowing down drastically, when Con-BN-SLIM takes over it is still able to find many improvements over the exact same networks. This demonstrates the notion of turbocharging quite well.

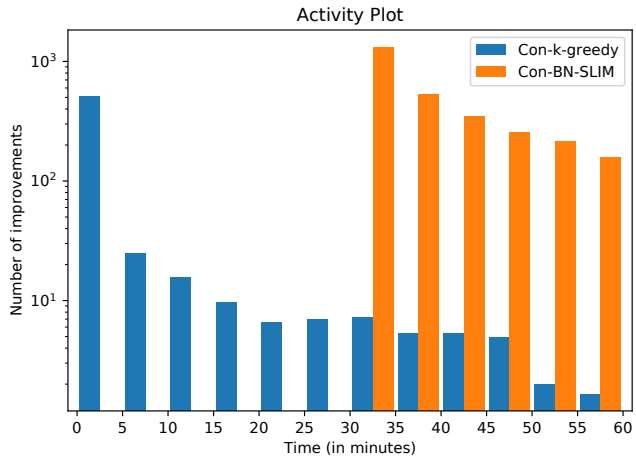

Figure 2: Activity plot showing rate of improvements of Con-BN-SLIM and Con-k-greedy against time. Note that the y-axis is in logscale.

Next, we compare the scores of the networks produced by Con-k-greedy and Con-BN-SLIM at the 60-minute mark. We use the ΔBIC metric to make this comparison. The difference in BIC scores of two networks approximates the ratio of their marginal likelihoods, which is the Bayes Factor [Raftery, 1995, Scanagatta et al., 2018]. The ΔBIC score of a pair of networks is mapped to a categorical scale, with positive scores signifying positive evidence towards the first network and vice versa. As can be seen from Table 4, Con-BN-SLIM severely outperforms Con-k-greedy.

Table 4: ΔBIC values comparing Con-BN-SLIM against Con-k-greedy

Category	Δ BIC	Count
extremely positive	(10, ∞)	132
strongly positive	(6, 10)	1
positive	(2, 6)	0
neutral	(-2, 2)	26
negative	(-6, -2)	0
strongly negative	(-10, -6)	1
extremely negative	($-\infty$, -10)	7

Finally, we compare the constraint satisfaction by the solutions of k-greedy, Con-k-greedy, and Con-BN-SLIM in Table 5. We measure and tabulate the percentage of total constraints satisfied. There are several noteworthy points here.

k-greedy We see that k-greedy, despite having no knowledge of the constraints manages to satisfy more than half of them. This could be attributed to the fact that k-greedy still has access to the score function caches whose job is to quantify and reflect the closeness of any network to the ground truth network (just like the expert constraints).

Table 5: Comparison of Constraint Satisfaction as a Percentage of Total Constraints

	Avg. % satisfied constraints		
Bucket	k-greedy	Con-k-greedy	Con-BN-SLIM
Small	77.74%	85.64%	90.85%
Medium	63.80%	74.39%	81.84%
Large	59.44%	89.30%	89.58%
All	67.54%	82.55%	87.23%

Con-k-greedy We see a clear improvement in the constraint satisfaction by Con-k-greedy in comparison to k-greedy. This is to be expected as we modified the heuristic to consider the expert constraints.

Con-BN-SLIM We see that Con-BN-SLIM ends up satisfying slightly more constraints than Con-k-greedy even though it was not intentionally designed to do so. This, however, is a favorable side-effect. Con-BN-SLIM never violates a constraint that was satisfied by the initial heuristic solution. Thus, by random chance, the number of satisfied constraints can only increase.

6 CONCLUSION

We have proposed the first method for BN structure learning that scales to large instances while respecting treewidth bounds and soft expert constraints. At the heart of our method is the utilization of a MaxSAT encoding, applied locally, which demonstrates the flexibility of the SLIM framework.

We see several possibilities for improving the portion of satisfied expert constraints. An easy target is to improve the Phase 1 heuristics to handle better the root bag constriction, which a MaxSAT encoding could provide. Even more potential might be to adapt other heuristics like k-MAX [Scanagatta et al., 2018] or Elimination Trees [Benjumeda et al., 2019] for Phase 1.

The current implementation does not actively try to increase the satisfied constraints in Phase 2. Somewhat surprising to us is to see still a significant increase. This suggests a learning approach, where we continuously check during Phase 2 whether any additional expert constraint is satisfied and, if so, add it as a hard constraint to the Phase 2 engine. This way, Phase 2 could yield a monotonic increase of the score and the number of satisfied constraints.

References

Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6): 716–723, 1974.

Marco Benjumeda, Concha Bielza, and Pedro Larrañaga. Learning tractable Bayesian networks in the space of elimination orders. *Artificial Intelligence*, 274:66–90, 2019.

Jeremias Berg, Matti Järvisalo, and Brandon M. Malone. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 86–95. JMLR.org, 2014.

Eunice Yuh-Jie Chen, Yujia Shen, Arthur Choi, and Adnan Darwiche. Learning Bayesian networks with ancestral constraints. *Advances in Neural Information Processing Systems*, 29, 2016.

David Maxwell Chickering. Learning equivalence classes of Bayesian network structures, 1996.

Jukka Corander, Tomi Janhunen, Jussi Rintanen, Henrik Nyman, and Johan Pensar. Learning Chordal Markov Networks by Constraint Satisfaction. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

Gal Elidan and Stephen Gould. Learning bounded treewidth Bayesian networks. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 417–424. Curran Associates, Inc., 2009.

Johannes K. Fichte, Neha Lodha, and Stefan Szeider. SAT-based local improvement for finding tree decompositions of small width. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 401–411. Springer Verlag, 2017. doi: 10.1007/978-3-319-66263-3_25.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

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

Russell J Kennett, Kevin B Korb, and Ann E Nicholson. Seabreeze prediction using bayesian networks. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 148–153. Springer, 2001.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: principles and techniques*. MIT press, 2009.

Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width Bayesian networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 370–378. JMLR.org, 2013.

Johan Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag. The necessity of bounded treewidth for efficient inference in Bayesian networks. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 237–242. IOS Press, 2010.

Andrew Li and Peter van Beek. Bayesian network structure learning with side constraints. In *International Conference on Probabilistic Graphical Models*, pages 225–236. PMLR, 2018.

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 179–195. Springer Verlag, 2016. doi: 10.1007/978-3-319-40970-2_12.

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. A SAT approach to branchwidth. *ACM Trans. Comput. Log.*, 20(3):15:1–15:24, 2019. doi: 10.1145/3326159.

Siqi Nie, Cassio Polpo de Campos, and Qiang Ji. Learning bounded tree-width Bayesian networks via sampling. In Sébastien Destercke and Thierry Denoeux, editors, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 13th European Conference, ECSQARU 2015, Compiègne, France, July 15-17, 2015. Proceedings*, volume 9161 of *Lecture Notes in Computer Science*, pages 387–396. Springer Verlag, 2015.

Pekka Parviainen, Hossein Shahrabi Farahani, and Jens Lagergren. Learning bounded tree-width Bayesian networks using Integer Linear Programming. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 751–759. JMLR.org, 2014.

Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Maxsat-based postprocessing for treedepth. In Helmut Simonis, editor, *Principles and Practice of Constraint Programming*, pages 478–495, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58475-7.

Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Turbocharging treewidth-bounded Bayesian network structure learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3895–3903, May 2021a.

Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. Learning fast-inference bayesian networks. *Advances in Neural Information Processing Systems*, 34, 2021b.

Vaidyanathan Peruvemba Ramaswamy and Stefan Szeider. aditya95sriram/bn-slim: Neurips 2021, October 2021c. URL `https://doi.org/10.5281/zenodo.5598257`.

Adrian E. Raftery. Bayesian model selection in social research. *Sociological Methodology*, 25:111–163, 1995. ISSN 00811750, 14679531.

Mauro Scanagatta. BLIP – Bayesian network learning and inference package, 2015. URL `https://ipg.idsia.ch/software/blip`.

Mauro Scanagatta, Giorgio Corani, Cassio Polpo de Campos, and Marco Zaffalon. Learning treewidth-bounded Bayesian networks with thousands of variables. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1462–1470, 2016.

Mauro Scanagatta, Giorgio Corani, Marco Zaffalon, Jaemin Yoo, and U Kang. Efficient learning of bounded-treewidth Bayesian networks from complete and incomplete data sets. *Int. J. Approx. Reason*, 95:152–166, 2018.

André Schidler and Stefan Szeider. SAT-based decision tree learning for large data sets. In *Proceedings of AAAI'21, the Thirty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, 2021.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.