Fast Bayesian Network Structure Learning using Quasi-Determinism Screening
Thibaud Rahier, Sylvain Marié, Stéphane Girard, Florence Forbes

To cite this version:
Thibaud Rahier, Sylvain Marié, Stéphane Girard, Florence Forbes. Fast Bayesian Network Structure Learning using Quasi-Determinism Screening. 2018. <hal-01691217v3>

HAL Id: hal-01691217
https://hal.archives-ouvertes.fr/hal-01691217v3
Submitted on 12 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Learning the structure of Bayesian networks from data is a NP-Hard problem that involves optimization over a super-exponential sized space. In this work, we show that in most real life datasets, a number of the arcs contained in the final structure can be pre-screened at low computational cost with a limited impact on the global graph score. We formalize the identification of these arcs via the notion of quasi-determinism, and propose an associated algorithm that narrows the structure learning task down to a subset of the original variables. We show, on diverse benchmark datasets, that this algorithm exhibits a significant decrease in computational time and complexity for only a little decrease in performance score.

1 INTRODUCTION

Bayesian networks are probabilistic graphical models that present interest both in terms of knowledge discovery and density estimation. Learning Bayesian networks from data has been however proven to be NP-Hard by Chickering (1996).

There has been extensive work on tackling the ambitious problem of Bayesian network structure learning from observational data. Algorithms fall under two main categories: constraint-based and score-based.

Constraint-based structure learning algorithms rely on testing for conditional independencies that hold in the data in order to reconstruct a Bayesian network encoding these independencies. The PC algorithm by Spirtes et al. (2000) was the first practical application of this idea, followed by several optimized approaches as the fast incremental association (Fast-IAMB) algorithm from Yaromakala and Margaritis (2005).

Score-based structure learning relies on the definition of a network score, then on the search for the best-scoring structure among all possible directed acyclic graphs (DAGs). The number of possible DAG structures with \( n \) nodes is of order \( 2^{O(n^2)} \), which prevents exhaustive search when \( n \) is typically larger than 30, even for the most recent algorithms Silander and Myllymaki (2012) or Yuan et al. (2013)).

Most of the score-based algorithms used in practice therefore rely on heuristics, as the original approach from Cooper and Herskovits (1992) which supposed a prior ordering of the variables to perform parent set selection, or Bouckaert (1995) who proposed to search through the structure space using greedy hill climbing with random restarts. Since these first algorithms, different approaches have been proposed: some based on the search for an optimal ordering as Chen et al. (2008) or Teyssier and Koller (2012), others on optimizing the search task in accordance to a given score (BIC or BDe) as de Campos and Ji (2011) or Scanagatta et al. (2015).

Meanwhile, data itself may contain determinism, for example in the fields of cancer risk identification (de Morais et al. (2008)) or nuclear safety (Mabrouk et al. (2014)). Moreover, data is increasingly collected and generated by software systems whether in social networks, smart buildings, smart grid, smart cities or the internet of things (IoT) in general (Koo et al. (2016)). These systems in their vast majority rely on relational data models or lately on semantic data models (El Kaed et al. (2016)) which cause deterministic relationships between variables to be more and more common in datasets. Determinism has been shown to interfere with Bayesian network structure learning, notably constraint-based methods as mentioned by Luo (2006).

In this paper, we focus on score-based algorithms. After reminding the background of Bayesian network structure learning in section 2, we state some theoretical results in section 3, that enable to bridge the gap between deter-
minimization and Bayesian network scoring.
In section 4, exploiting the intuition brought by these theoretical results, we propose and study the complexity of the quasi-deterministic screening algorithm. The idea is that some of the arcs contained in the desired Bayesian network can be learned during a quick screening phase where quasi-deterministic relationships are detected, thus narrowing the learning task down to a subset of the original variables.

In practice, not only does this algorithm accelerate the overall learning procedure with very low performance loss, but it also leads to sparser and therefore more interpretable graphs than state of the art methods, as presented using benchmark datasets in section 5.

Finally, section 6 is dedicated to a discussion and to numerous perspectives emerging from this work. Proofs of all lemmas and propositions are available in the supplementary material.

2 BAYESIAN NETWORK STRUCTURE LEARNING

2.1 Bayesian networks

Let \( X = (X_1, \ldots, X_n) \) be a \( n \)-tuple of categorical random variables with respective value sets \( \text{Val}(X_1), \ldots, \text{Val}(X_n) \). The distribution of \( X \) is denoted by, \( \forall x = (x_1, \ldots, x_n) \in \text{Val}(X) \),

\[
p(x) = P(X_1 = x_1, \ldots, X_n = x_n).
\]

For \( I \subseteq [1, n] \), we define \( X_I = \{X_i\}_{i \in I} \), and the notation \( p(\cdot) \) and \( p(\cdot|\cdot) \) is extended to the marginals and conditionals of any subset of variables: \( \forall (x_I, x_J) \in \text{Val}(X_I) \times \text{Val}(X_J), p(x_I|x_J) = P(X_I = x_I|X_J = x_J) \).

Moreover, we suppose that \( D \) is a dataset containing \( M \) i.i.d. instances of \( (X_1, \ldots, X_n) \). All quantities empirically computed from \( D \) will be written with a \( \hat{\cdot} \) exponent (e.g. \( \hat{p} \) refers to the empirical distribution with respect to \( D \)). Finally, \( D_I \) refers to the restriction of \( D \) to the observations of \( X_I \).

A Bayesian network is an object \( \mathcal{B} = (G, \theta) \) where

- \( G = (V, A) \) is a directed acyclic graph (DAG) structure with \( V \) the set of nodes and \( A \subseteq V \times V \) the set of arcs. We suppose \( V = [1, n] \) where each node \( i \in V \) is associated with the random variable \( X_i \), and \( \pi^G(i) = \{ j \in V \text{ s.t. } (j, i) \in A \} \) is the set of \( i \)'s parents in \( G \). The exponent \( G \) may be dropped for clarity when the referred graph is obvious.
- \( \theta = \{ \theta_i \}_{i \in V} \) is a set of parameters. Each \( \theta_i \) defines the local conditional distribution \( P(X_i|X_{\pi(i)}) \).

More precisely, \( \theta_i = \{ \theta_{x_i|\pi(i)} \} \) where for \( i \in V, x_i \in \text{Val}(X_i) \) and \( \pi(i) \in \text{Val}(X_{\pi(i)}) \),

\[
\theta_{x_i|\pi(i)} = p(x_i|\pi(i)).
\]

A Bayesian network \( \mathcal{B} = (G, \theta) \) encodes the following factorization of the distribution of \( X \): for \( x = (x_1, \ldots, x_n) \in \text{Val}(X) \),

\[
p(x) = \prod_{i=1}^n p(x_i|\pi(i)) = \prod_{i=1}^n \theta_{x_i|\pi(i)}.
\]

Such a factorization notably implies that each variable is independent of its non-descendants given its parents.

2.2 Score-based approach to Bayesian network structure learning

Suppose we have a scoring function \( s : \text{DAG}_{\mathcal{V}} \rightarrow \mathbb{R} \), where \( \text{DAG}_{\mathcal{V}} \) is the set of all possible DAG structures with node set \( \mathcal{V} \). Score-based Bayesian network structure learning comes down to solving the following combinatorial optimization problem:

\[
G^* \in \arg\max_{G \in \text{DAG}_{\mathcal{V}}} s(G). \tag{1}
\]

It can be shown that \( 2^{2(n-1)} \leq |\text{DAG}_{\mathcal{V}}| \leq 2^{n(n-1)} \) where \(|\mathcal{V}| = n \). There are therefore \( 2^{O(n^2)} \) possible DAG structures containing \( n \) nodes; the size of \( \text{DAG}_{\mathcal{V}} \) is said to be super-exponential in \(|\mathcal{V}|\).

Most scoring functions used in practice are based on the likelihood function. The most straightforward being the Max log-likelihood score, that we now present.

The Max log-likelihood score Let \( l(\theta : D) = \log(\hat{p}_D(\theta)) \) be the log-likelihood of the set of parameters \( \theta \) given the dataset \( D \).

For a given DAG structure \( G \in \text{DAG}_{\mathcal{V}} \), we define the Max log-likelihood (MLL) score of \( G \) associated with a dataset \( D \) as:

\[
s^{\text{MLL}}(G : D) = \max_{\theta \in \Theta_G} l(\theta : D),
\]

where \( \Theta_G \) is the set of all \( \theta \)'s such that \( \mathcal{B} = (G, \theta) \) is a Bayesian network.

The MLL score is very straightforward and intuitive, but it favorsizes denser structures: if \( G_1 = (V, A_1) \) and \( G_2 = (V, A_2) \) are two graph structures such that \( A_1 \subseteq A_2 \), we can show that: \( s^{\text{MLL}}(G_1 : D) \leq s^{\text{MLL}}(G_2 : D) \).

There are two main (non-exclusive) approaches to solve this problem:

- Constrain the structure space to avoid learning overly complex graphs, which is the idea of hybrid structure learning algorithms such as the sparse candidate algorithm presented by [Friedman et al., 1999], or the Max-Min Hill Climbing (MMHC) algorithm introduced by [Tsamardinos et al., 2006].
• Use a score that induces a goodness-of-fit versus complexity tradeoff, such as:
  - BIC (Schwarz et al. (1978)), which is a penalized version of the MLL score.
  - BDe (Heckerman et al. (1995)), which is derived from the marginalization of the likelihood, implying a prior implicitly penalizing the model’s complexity.

The BDe score In this paper, we will use the BDe score to evaluate a BN structure’s quality, as it is done in several recent papers as Teyssier and Koller (2012), or Nie et al. (2016). This score is known to be a good indicator of generalization performance.

The BDe score of $G \in \text{DAG}_V$ is defined as the log of the marginal likelihood, integrated against a Dirichlet prior. Assuming a uniform prior over all network structures, we have

$$s^{\text{BDe}}(G : D) = \log \left( \int_{\theta \in \Theta_G} p(D|\theta, G) \frac{p(\theta|G)}{\text{Dirichlet prior}} d\theta \right)$$

In practice, we often use the BDeu score, introduced by Buntine (1991), which is a particular case of BDe where the Dirichlet prior assigns the same probability to all configurations of $X_i \cup X_{\pi(i)}$ for each $i$.

3 DETERMINISM AND BAYESIAN NETWORKS

3.1 Definitions

We propose the following definitions of determinism and deterministic DAGs using the notion of conditional entropy. In this paper, determinism will always be meant empirically, with respect to a dataset $D$.

Definition 1 Determinism wrt $D$
Given a dataset $D$ containing observations of variables $X_i$ and $X_j$, the relationship $X_i \rightarrow X_j$ is deterministic with respect to $D$ iff $H^D(X_i|X_j) = 0$, where

$$H^D(X_i|X_j) = -\sum_{x_i,x_j} p^D(x_i, x_j) \log(p^D(x_i|x_j))$$

is the empirical conditional Shannon entropy.

It is straightforward to prove that Definition 1 relates to a common and intuitive perception of determinism, as the one presented by Luo (2006). Indeed,

$$H^D(X_i|X_j) = 0 \iff \forall x_j \in \text{Val}(X_j), \exists! x_i \in \text{Val}(X_i) \text{ s.t. } p^D(x_i|x_j) = 1.$$

This definition is naturally extended to $X_I$ and $X_J$ for $I, J \subseteq V$, i.e. $X_I \rightarrow X_J$ is deterministic with respect to $D$ iff $H^D(X_J|X_I) = 0$.

Definition 2 Deterministic DAG wrt $D$ $G \in \text{DAG}_V$ is said to be deterministic with respect to $D$ iff $\forall v \in V$ s.t. $\pi^G(v) \neq \emptyset$, $X_{\pi^G(v)} \rightarrow X_v$ is deterministic wrt $D$.

3.2 Deterministic trees and MLL score

We first recall a lemma that relates the MLL score presented in section 2 to the notion of empirical conditional entropy. This result is well known and notably stated by Koller and Friedman (2009).

Lemma 1 For $G \in \text{DAG}_V$ associated with variables $X_1, \ldots, X_n$ observed in a dataset $D$,

$$s^{\text{MLL}}(G : D) = -M \sum_{i=1}^n H^D(X_i|X_{\pi(i)})$$

where by convention $H^D(X_i|X_{\emptyset}) = H^D(X_i)$.

The next proposition follows then straightforwardly. We remind that a tree is a DAG in which each node has exactly one parent, except for the root node which has none.

Proposition 1 If $T$ is a deterministic tree with respect to $D$, then $T$ is a solution of $\{1\}$:

$$s^{\text{MLL}}(T : D) = \max_{G \in \text{DAG}_V} s^{\text{MLL}}(G : D).$$

It is worth noticing that complete DAGs also maximize the MLL score. The main interest of Proposition 1 resides in the fact that, under the (strong) assumption that a deterministic tree exists, we are able to explicit a sparse solution of $\{1\}$, with $n-1$ arcs instead of $\frac{n(n-1)}{2}$ for a complete DAG.

3.3 Deterministic forests and the MLL score

The deterministic tree assumption of Proposition 1 is very restrictive. In this section, it is extended to deterministic forests, defined as follows:

Definition 3 Deterministic forest wrt $D$ $F \in \text{DAG}_V$ is said to be a deterministic forest with respect to $D$ iff $F = \bigcup_{k=1}^p T_k$, where $T_1, \ldots, T_p$ are $p$ disjoint deterministic trees wrt $D_{V_{T_1}}, \ldots, D_{V_{T_p}}$ respectively and s.t. $\bigcup_{k=1}^p V_{T_k} = V$.

In the expression $\bigcup_{k=1}^p T_k$, $\cup$ is the canonical union for graphs: $G \cup G' = (V_G \cup V_{G'}, A_G \cup A_{G'})$. 
For a given deterministic forest \( F \) wrt \( D \), we define
\[
R(F) = \{ i \in V \mid \pi^F(i) = \emptyset \}
\]
the set of \( F \)'s roots (the union of the roots of each of its trees).

**Proposition 2** Suppose \( F \) is a deterministic forest wrt \( D \). Let \( G^*_R(F) \) be a solution of the structure learning optimization problem \( \mathcal{P} \) for \( X_R(F) \) and the MLL score i.e.
\[
s^{MLL}(G^*_R(F) : D_R(F)) = \max_{G \in DAG_R(F)} s^{MLL}(G : D_R(F)).
\]
Then, \( G^* = F \cup G^*_R(F) \) is a solution of \( \mathcal{P} \) for \( X \), i.e.
\[
s^{MLL}(G^* : D) = \max_{G \in DAG_V} s^{MLL}(G : D).
\]

As opposed to Proposition 1, the assumptions of Proposition 2 are always formally verified: if there is no determinism in the dataset \( D \), then \( R(F) = V \), and every tree \( T_k \) is formed of a single root node. In that case, solving problem \( \mathcal{P} \) for \( G^*_R(F) \) is the same as solving it for \( G^* \). Of course, we are interested in the case where \( R(F) < n \), as this enables us to focus on a smaller structure learning problem while still having the guarantee to learn the optimal Bayesian network with regards to the MLL score.

As seen in section 2, the main issue with the MLL score is that it favors complete graphs. However, a deterministic forest \( F \) containing \( p \) trees is very sparse \((n - p \text{ arcs})\), and even if the graph \( G^*_R(F) \) is dense, the graph \( G^* = F \cup G^*_R(F) \) may still satisfy sparsity conditions.

### 4 Structure Learning with Quasi-determinism Screening

#### 4.1 Quasi-determinism

When it comes to Bayesian network structure learning algorithms, even heuristics are computationally intensive. We would like to use the theoretical results presented in section 3 to simplify the structure learning problem.

Our idea is to narrow the structure learning problem down to a subset of the original variables: the roots of a deterministic forest, in order to significantly decrease the overall computation time. This is what we call deterministic screening.

However, in many datasets, one does not observe real empirical determinism, although there are very strong relationships between some of the variables. We therefore propose to relax the aforementioned determinism screening to quasi-determinism screening, where *quasi* is meant with respect to a parameter \( \epsilon \); we talk about \( \epsilon \)-quasi-determinism.

There are several ways to measure how close a relationship is from deterministic. [Huhtala et al. (1999)](Huhtala et al. (1999)) consider the minimum number of observations that must be dropped from the data for the relationship to be deterministic. Since we are in a score-maximization context, we will rather use \( \epsilon \) as a threshold on the empirical conditional entropy. The following definition is the natural generalization of Definition 1.

**Definition 4** \( \epsilon \)-quasi-determinism \((\epsilon \text{-qd})\)

Given a dataset \( D \) containing observations of variables \( X_i \) and \( X_j \), the relationship \( X_i \rightarrow X_j \) is \( \epsilon \text{-qd} \) wrt \( D \) iff
\[
H^D(X_j | X_i) \leq \epsilon.
\]

It has been seen in Proposition 2 that a deterministic forest is the subgraph of an optimal DAG with respect to the MLL score, while still satisfying sparsity conditions. Such a forest is therefore very promising with regards to the fit-complexity tradeoff (typically evaluated by scores such as BDe or BIC).

Combining this intuition with the \( \epsilon \text{-qd} \) criteria presented in Definition 4, we propose the quasi-determinism screening approach to Bayesian network structure learning, defined in the next subsections.

#### 4.2 Quasi-determinism screening algorithm

Algorithm 1 details how to find the simplest \( \epsilon \text{-qd} \) forest \( F_\epsilon \) from a dataset \( D \) and a threshold \( \epsilon \). Here *simplest* refers to the complexity in terms of number of parameters.

This algorithm takes for input:
- \( D \): a dataset containing \( M \) observations of \( X \),
- \( \epsilon \): a threshold for quasi-determinism.

The next proposition states that Algorithm 1 is well defined.

**Proposition 3** For any rightful input \( D \) and \( \epsilon \), the output of Algorithm 1 is a forest (i.e. a directed acyclic graph with at most one parent per node).

#### 4.3 Learning Bayesian networks using quasi-determinism screening

We now present Algorithm 2, which uses quasi-determinism screening to accelerate Bayesian network structure learning. This algorithm takes the following input:
Algorithm 1 Quasi-determinism screening (qds)

Input: \( D, \epsilon \)

1. Compute empirical conditional entropy matrix \( H^D = (H^D(X_i, X_j))_{1 \leq i, j \leq n} \)
2. for \( i = 1 \) to \( n \) do #identify the set of potential \( \epsilon \)–qds parents for each \( i \)
   3. compute \( \pi^*_i(i) = \{ j \in [1, n] \setminus \{ i \} \mid H^D_{ij} \leq \epsilon \} \)
4. for \( i = 1 \) to \( n \) do #check for cycles in \( \epsilon \)–qds relations
   5. if \( \exists j \in \pi^*_i(i) \) s.t. \( i \in \pi^*_j(j) \) then
      6. if \( H^D_{ji} \leq H^D_{ij} \) then
         7. \( \pi^*_i(j) \leftarrow \pi^*_i(j) \setminus \{ i \} \)
   8. else
      9. \( \pi^*_i(i) \leftarrow \pi^*_i(i) \) \( \setminus \{ j \} \)
10. for \( i = 1 \) to \( n \) do #choose the simplest among all potential parents
   11. \( \pi^*_i(i) \leftarrow \arg\min_{j \in \pi^*_i(i)} |Val(X_j)| \)
12. Compute forest \( F_\epsilon = (V_{F_\epsilon}, A_{F_\epsilon}) \) where \( V_{F_\epsilon} = [1, n] \) and \( A_{F_\epsilon} = \{(\pi^*_i(i), i) \mid i \in [1, n], s.t. \pi^*_i(i) \neq \emptyset\} \)

Output: \( F_\epsilon \)

- \( D \): a dataset containing \( M \) observations of \( X \),
- \( \epsilon \): a threshold for quasi-determinism,
- sota-BNSL: a state of the art structure learning algorithm, taking for input a dataset, and returning a Bayesian network structure.

Algorithm 2 Bayesian network structure learning with quasi deterministic screening (qds-BNSL)

Input: \( D, \epsilon, \text{sota-BNSL} \)

1. Compute \( F_\epsilon \) by running Algorithm 1 with input \( D \) and \( \epsilon \)
2. Identify \( R(F_\epsilon) = \{ i \in [1, n] \mid \pi^{F_\epsilon}(i) = \emptyset \} \), the set of \( F_\epsilon \)’s roots.
3. Compute \( G^*_R(F_\epsilon) \) by running sota-BNSL on \( D_{R(F_\epsilon)} \)
4. \( G^*_\epsilon \leftarrow F_\epsilon \cup G^*_R(F_\epsilon) \)

Output: \( G^*_\epsilon \)

The extension of the definition of determinism to quasi-determinism (Definition 3) prevents us to have ‘hard’ guarantees as those presented in Proposition 2. However, we are able to explicit bounds for the MLL score of a graph \( G^*_\epsilon \) returned by Algorithm 2, as stated in the following Proposition.

**Proposition 4** Let \( \epsilon, D \) and sota-BNSL be rightful input to Algorithm 2, and \( G^*_\epsilon \) the associated output.
Then, if sota-BNSL is exact (i.e. always returns an optimal solution) with respect to the MLL score, we have the following lower bound for \( s^\text{MLL}(G^*_\epsilon : D) \):

\[
s^\text{MLL}(G^*_\epsilon : D) \geq \left( \max_{G \in \text{DAG}_V} s^\text{MLL}(G : D) \right) - Mnc.
\]

In practice, this bound is not very tight and this result therefore has few applicative potential. However, it shows that:

\[
s^\text{MLL}(G^*_\epsilon : D) \to \max_{G \in \text{DAG}_V} s^\text{MLL}(G : D).
\]

In other words, \( \epsilon \mapsto s^\text{MLL}(G^*_\epsilon : D) \) is continuous in \( \epsilon \), and Proposition 4 generalizes Proposition 2.

Algorithm 2 is promising, notably if for small \( \epsilon \) we can significantly decrease the number of variables to be considered by sota-BNSL. We would in this case have a much faster algorithm with a controlled performance loss.

### 4.4 Complexity analysis

**Complexity of the state of the art algorithm** The number of possible DAG structures being super exponential in the number of nodes, state of the art algorithms do not entirely explore the structure space but use smart caching and pruning methods to have a good performance & computation time trade-off.

Let sota-BNSL be a state of the art Bayesian network structure learning algorithm and \( C_{\text{sota}}(M, n) \) be its complexity.

\( C_{\text{sota}}(M, n) \) should typically be thought of as linear in \( M \) and exponential, or at least high degree polynomial, in \( n \) for the best algorithms.

**Complexity of Algorithm 1** We have the following decomposition of the complexity of Algorithm 1:

1. Lines 1-3: \( O(Mn^2) \). Computation of \( H^D \): we need counts for every couple \((X_i, X_j)\) for \( i < j \) (each time going through all rows of \( D \)), which implies \( M \frac{2(n(n-1))}{2} \) operations.
2. Lines 4-9: \( O(n^2) \). Going through all elements of \( H^D \) once.
3. Lines 10-12: \( O(n^2) \). Going through all elements of \( H^D \) once.

Overall one has that \( C_{\text{Alg1}}(M, n) = O(Mn^2) \).

**Complexity of Algorithm 2** For a given dataset \( D \), we define:

\[
\forall \epsilon \geq 0, \ n_\epsilon(\epsilon) = |R(F_\epsilon)|.
\]

The function \( n_\epsilon(\cdot) \), associates to \( \epsilon \geq 0 \) the number of roots of the forest \( F_\epsilon \) returned by Algorithm 1. The complexity of Algorithm 2 then decomposes as:
1. Line 1: $O(Mn^2)$. Run of Algorithm 1.

2. Lines 2-4: $C_{sota}(M, n_r(\epsilon))$. Run of sota-BNSL on reduced dataset $D_{R(F)}$ with $n_r(\epsilon)$ columns.

This yields $C_{Alg2}(M, n) = O(Mn^2) + C_{sota}(M, n_r(\epsilon))$.

We are interested in how much it differs from $C_{sota}(M, n)$, which depends mainly on:

- how $n_r(\epsilon)$ compares to $n$,
- how $C_{sota}(M, n)$ varies with respect to $n$.

It is hard to obtain a theoretical difference $C_{sota} - C_{Alg2}$, since it is not clear how to estimate the complexity of state of the art learning algorithms, often based on local search heuristics. However, we know that all Bayesian network structure learning algorithms are very time-intensive, which lets us expect an important decrease in computational time for Algorithm 2 compared to a state of the art algorithm.

In the next section, we run a state of the art structure learning algorithm and Algorithm 2 on benchmark datasets in order to confirm this intuition.

## 5 EXPERIMENTS

### 5.1 Experimental setup

**Data** Table 1 summarizes the data used in our experiments. We considered the largest open-source categorical datasets among those presented by Davis and Domingos (2010) and available on the UCI repository (Dheeru and Karra Taniskidou (2017)): 20 Newsgroup, Adult, Book, Covertype, KDDCup 2000, MSWeb, Plants, Reuters-52 and USCensus. Moreover, as it was done by Scanagatta et al. (2016), we chose the largest Bayesian networks available in the literature for each of which we simulated 10000 observations: Andes, Hailfinder, Hepar 2, Link, Munin 1-4, Pathfinder and Win95pts.

**Programming details and choice of sota-BNSL**

Most of the code associated with this project was done in R, enabling an optimal exploitation of the bnlearn package from Scutari (2009), which is a very good reference among open-source packages dealing with Bayesian networks structure learning.

We need a state of the art Bayesian network structure learning algorithm to obtain a baseline performance. After carefully evaluating several algorithms implemented in the bnlearn package, we chose to use Greedy Hill Climbing with random restarts and a tabu list, as it consistently outperformed other built-in algorithms both in time and score, in addition to being also used as a benchmark algorithm in the literature, notably by Teyssier and Koller (2012). In this section, we refer to this algorithm as sota-BNSL.

**Choice of $c$ for qds-BNSL**

An approach to choosing $c$ in the case of the qds-BNSL algorithm is to pick values for $n_r(\epsilon)$, and manually find the corresponding values for $c$. For a given dataset and $x \in [0, 1]$, we define $x = n_r(\epsilon)$ for which the number of roots of the qd forest $F_q$ represents a proportion $x$ of the total number of variables.

The computation of $x$ is not problematic: once $H^D$ is computed and stored, evaluating $n_r(\epsilon)$ is done in constant time, and finding one of $n_r(\epsilon)$’s quantiles is doable in $O(\log(n))$ operations (dichotomy), which is negligible compared to the overall complexity of the screening.

**Algorithm evaluation** The algorithms are evaluated using 3 axes of performance:

- Bayesian network BDeu score presented in section 2 with equivalent sample size (ESS) equal to 5, inspired from Teyssier and Koller (2012).
- Number of arcs of the learned Bayesian network. The BDeu score naturally penalizes overly complex models (in terms of number of parameters), it is

### Table 1: Datasets presentation

<table>
<thead>
<tr>
<th>name</th>
<th>short name</th>
<th>$n$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 newsgroups</td>
<td>20ng</td>
<td>930</td>
<td>11293</td>
</tr>
<tr>
<td>adult</td>
<td>adult</td>
<td>125</td>
<td>36631</td>
</tr>
<tr>
<td>book</td>
<td>book</td>
<td>500</td>
<td>8700</td>
</tr>
<tr>
<td>covertype</td>
<td>covertype</td>
<td>84</td>
<td>30000</td>
</tr>
<tr>
<td>kddcup 2000</td>
<td>kddcup</td>
<td>64</td>
<td>180092</td>
</tr>
<tr>
<td>msnbc</td>
<td>msnbc</td>
<td>17</td>
<td>291326</td>
</tr>
<tr>
<td>msweb</td>
<td>msweb</td>
<td>294</td>
<td>29441</td>
</tr>
<tr>
<td>plants</td>
<td>plants</td>
<td>69</td>
<td>17412</td>
</tr>
<tr>
<td>reuters 52</td>
<td>r52</td>
<td>941</td>
<td>6532</td>
</tr>
<tr>
<td>uscensus</td>
<td>uscensus</td>
<td>68</td>
<td>245825</td>
</tr>
<tr>
<td>andes</td>
<td>andes</td>
<td>223</td>
<td>10000</td>
</tr>
<tr>
<td>hailfinder</td>
<td>hailfinder</td>
<td>56</td>
<td>10000</td>
</tr>
<tr>
<td>hepar 2</td>
<td>hepar2</td>
<td>70</td>
<td>10000</td>
</tr>
<tr>
<td>link</td>
<td>link</td>
<td>724</td>
<td>10000</td>
</tr>
<tr>
<td>munin 1</td>
<td>munin1</td>
<td>186</td>
<td>10000</td>
</tr>
<tr>
<td>munin 2</td>
<td>munin2</td>
<td>1003</td>
<td>10000</td>
</tr>
<tr>
<td>munin 3</td>
<td>munin3</td>
<td>1041</td>
<td>10000</td>
</tr>
<tr>
<td>munin 4</td>
<td>munin4</td>
<td>1038</td>
<td>10000</td>
</tr>
<tr>
<td>pathfinder</td>
<td>pathfinder</td>
<td>109</td>
<td>10000</td>
</tr>
<tr>
<td>windows 95 pts</td>
<td>win95pts</td>
<td>76</td>
<td>10000</td>
</tr>
</tbody>
</table>
however interesting to look at the number of arcs, as it is a straightforward way to evaluate how complex a Bayesian network structure appears to a human expert (and therefore how interpretable this structure is).

- Computing time $t_{run}$ (all algorithms were run on the same machine).

It is essential to remark that sota-BNSL is used both to obtain a baseline performance and inside qds-BNSL. In both cases, it is run with the same settings until convergence. The comparison of computing times is therefore fair.

We present the obtained results for our selected state of the art algorithm sota-BNSL, and 3 versions of qds-BNSL. For each dataset, we selected $\epsilon \in \{0.9, 0.95, 0.98\}$, corresponding to a reduction of sota-BNSL to 90%, 75% and 50% of the original variables respectively.

The results are shown in Tables 2-4, one per evaluation criterion. In each table, the actual value of the criterion is displayed for sota-BNSL (sota), and the relative difference is displayed for the three versions of qds-BNSL we consider ($\text{qds}_{\epsilon=0.9}$, $\text{qds}_{\epsilon=0.95}$ and $\text{qds}_{\epsilon=0.98}$).

### 5.2 Results

#### Score

It appears in Table 2 that the decrease in BDeu score is smaller than 5% for all the considered datasets when 90% of the variables remain after the pre-screening ($\text{qds}_{\epsilon=0.9}$), and for most of them when 75% of the variables remain ($\text{qds}_{\epsilon=0.95}$). This is also observed with $\epsilon=0.5$ for datasets that contain a lot of very strong pairwise relationships as kddcup, msweb, or munin2-4.

#### Computing time

Table 3 shows a significant decrease in computational time for qds-BNSL, which is all the more important that $\epsilon$ is large. In the best cases, we have both a very small decrease in BDeu score, and an important decrease in computational time. For example, the algorithm qds-BNSL with $\epsilon=0.5$ is 55% faster for msweb, and 54% for munin 3, while implying only around 1% decrease in score compared to sota-BNSL. If we allow a 5% score decrease, qds-BNSL can be up to 70% faster (20 newsgroups, book, msnbc, kddcup, hepar2, pathfinder).

These results confirm the complexity analysis of the previous section, in which we supposed that the screening phase had a very small computational cost compared to the standard structure learning phase.

#### Complexity

As showed by Table 4, Bayesian networks learnt with qds-BNSL are consistently less complex than those learnt with sota-BNSL. Several graphs learnt with $\text{qds}_{\epsilon=0.9}$ are more than 30% sparser while still scoring less than 5% below state of the art: 20 newsgroups, book, kddcup 2000, msnbc, msweb and hepar 2.

Figure 1 and 2 display two Bayesian networks learnt on the ‘msnbc’ dataset. They provide an interesting example of the sparsity induced by qds-BNSL. After the qds screening phase, half of the variables (corresponding to the nodes in white) are considered to be sufficiently explained by V1. They are therefore not taken into account by sota-BNSL, which is run only on the variables corresponding to the nodes in gray.

In the case of msnbc, this restriction of the learning problem implies only a small decrease in the final graph’s generalization performance (as reflected by the BDeu scores), while being 7 times faster to compute and enabling a significantly better readability.

In this processed version of the msnbc dataset (Davis and Domingos (2010)), each variable contains a binary information regarding the visit of a given page from the msnbc.com website. The Bayesian network displayed in Figure 2 shows in a compact way the influence be-

---

Table 2: BDeu score averaged by observation. Every result that is less than 5% smaller than sota-BNSL’s score is boldfaced.

<table>
<thead>
<tr>
<th>dataset</th>
<th>sota</th>
<th>$\text{qds}_{\epsilon=0.9}$ (%)</th>
<th>$\text{qds}_{\epsilon=0.95}$ (%)</th>
<th>$\text{qds}_{\epsilon=0.98}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20ng</td>
<td>-142.71</td>
<td>-0.66</td>
<td>-2.13</td>
<td>-4.78</td>
</tr>
<tr>
<td>adult</td>
<td>-12.86</td>
<td>-0.16</td>
<td>-0.05</td>
<td>-4.01</td>
</tr>
<tr>
<td>book</td>
<td>-34.81</td>
<td>-0.80</td>
<td>-1.69</td>
<td>-4.64</td>
</tr>
<tr>
<td>covertype</td>
<td>-13.60</td>
<td>-0.81</td>
<td>-1.23</td>
<td>-11.7</td>
</tr>
<tr>
<td>kddcup</td>
<td>-2.38</td>
<td>-0.31</td>
<td>-1.04</td>
<td>-3.83</td>
</tr>
<tr>
<td>msnbc</td>
<td>-6.19</td>
<td>-0.14</td>
<td>-2.62</td>
<td>-4.64</td>
</tr>
<tr>
<td>msweb</td>
<td>-9.77</td>
<td>+0.03</td>
<td>-0.07</td>
<td>-0.99</td>
</tr>
<tr>
<td>plants</td>
<td>-13.03</td>
<td>-2.57</td>
<td>-7.56</td>
<td>-20.92</td>
</tr>
<tr>
<td>r52</td>
<td>-95.48</td>
<td>-0.76</td>
<td>-1.96</td>
<td>-6.11</td>
</tr>
<tr>
<td>uscensus</td>
<td>-23.20</td>
<td>-0.27</td>
<td>-1.75</td>
<td>-10.39</td>
</tr>
<tr>
<td>andes</td>
<td>-93.23</td>
<td>-0.49</td>
<td>-0.22</td>
<td>-16.57</td>
</tr>
<tr>
<td>halffinder</td>
<td>-49.63</td>
<td>-0.06</td>
<td>-2.71</td>
<td>-10.21</td>
</tr>
<tr>
<td>hepar2</td>
<td>-32.60</td>
<td>-0.28</td>
<td>-1.36</td>
<td>-3.22</td>
</tr>
<tr>
<td>link</td>
<td>-215.68</td>
<td>+0.10</td>
<td>+1.10</td>
<td>-16.99</td>
</tr>
<tr>
<td>munin1</td>
<td>-41.15</td>
<td>-0.09</td>
<td>-0.16</td>
<td>-9.95</td>
</tr>
<tr>
<td>munin2</td>
<td>-171.82</td>
<td>-0.02</td>
<td>-0.02</td>
<td>-1.83</td>
</tr>
<tr>
<td>munin3</td>
<td>-165.09</td>
<td>0.00</td>
<td>0.00</td>
<td>-1.10</td>
</tr>
<tr>
<td>munin4</td>
<td>-186.11</td>
<td>-0.02</td>
<td>-0.02</td>
<td>-3.86</td>
</tr>
<tr>
<td>pathfinder</td>
<td>-26.65</td>
<td>-0.66</td>
<td>-0.70</td>
<td>-4.88</td>
</tr>
<tr>
<td>win95pts</td>
<td>-9.22</td>
<td>+0.06</td>
<td>-1.08</td>
<td>-9.15</td>
</tr>
</tbody>
</table>
between the different variables. For instance, we see that visits of the website’s pages corresponding to nodes in white (e.g. ‘weather’ (V8), ‘health’ (V9) or ‘business’ (V11)) are importantly influenced by whether the user has also visited the frontpage (V1). For example, learnt parameters show that a user who did not visit the website’s frontpage (V1) is about 10 times more likely to have visited the website’s ‘summary’ page (V13) than a user who did visit the frontpage. Such information is much harder to read from the graph learnt with sota-BNSL displayed in Figure 1.

6 DISCUSSION

We have seen that, both in theory and in practice, the quasi-determinism screening approach enables a decrease in computational time and complexity for a small decrease in graph score. This tradeoff is all the more advantageous that there actually are strong pairwise relationships in the data, that can be detected during the screening phase, thus enabling a decrease in the number of variables to be considered by the state of the art structure learning algorithm during the second phase of Algorithm 2.

Optimal cases for this algorithm take place when \( n_{r}(\epsilon) \) is significantly smaller than \( n \) for \( \epsilon \) reasonably small compared to the variable’s entropies. In practice this is reasonably frequent (e.g 20 newsgroup, msnbc, munion2-4, webkb among others).

Experiments presented in this paper were only conducted on opensource datasets that are frequently used by the community. However, we have also tested our algorithm on industrial descriptive metadatasets from the IoT domain, for which many variables possess (empirically) deterministic parents: we have \( n_{r}(\epsilon = 0) \) very small with respect to \( n \) (typically \( n_{r}(\epsilon = 0) \approx 10 \) for \( n \approx 100 \)). In this context qds-BNSL is up to 20 times faster than sota-BNSL, with equivalent or even better learned graphs in terms of BDeu score. This is a specific case (data stored in relational databases in almost perfect third normal form), but it is still interesting since an important amount of data accessible today contains determinism, as previously noted.

Besides, we still have potential to improve the qds-BNSL algorithm, by paralellizing the computation of \( H^{P} \), and implementing it in C instead of R.

Our main research perspective is to be able to anticipate

<table>
<thead>
<tr>
<th>dataset</th>
<th>sota (seconds)</th>
<th>qds_{0.9} (%)</th>
<th>qds_{0.75} (%)</th>
<th>qds_{0.5} (%)</th>
<th>dataset</th>
<th>sota</th>
<th>qds_{0.9} (%)</th>
<th>qds_{0.75} (%)</th>
<th>qds_{0.5} (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20ng</td>
<td>21,495</td>
<td>-1.62</td>
<td>-42.66</td>
<td>-72.94</td>
<td>20ng</td>
<td>3136</td>
<td>-4.50</td>
<td>-14.89</td>
<td>-31.89</td>
</tr>
<tr>
<td>adult</td>
<td>1,02</td>
<td>-6.61</td>
<td>-22.03</td>
<td>-61.20</td>
<td>adult</td>
<td>371</td>
<td>3.23</td>
<td>7.01</td>
<td>-13.75</td>
</tr>
<tr>
<td>covertype</td>
<td>565</td>
<td>-6.80</td>
<td>-33.22</td>
<td>-71.13</td>
<td>covertype</td>
<td>337</td>
<td>-0.89</td>
<td>-11.28</td>
<td>-37.69</td>
</tr>
<tr>
<td>msnbc</td>
<td>252</td>
<td>-20.66</td>
<td>-60.61</td>
<td>-85.65</td>
<td>msnbc</td>
<td>102</td>
<td>-7.84</td>
<td>-33.33</td>
<td>-63.73</td>
</tr>
<tr>
<td>msweb</td>
<td>4,701</td>
<td>-6.29</td>
<td>-9.86</td>
<td>-55.08</td>
<td>msweb</td>
<td>1,264</td>
<td>-2.53</td>
<td>-3.56</td>
<td>-34.97</td>
</tr>
<tr>
<td>plants</td>
<td>455</td>
<td>-46.93</td>
<td>-61.93</td>
<td>-84.07</td>
<td>plants</td>
<td>320</td>
<td>-6.25</td>
<td>-18.44</td>
<td>-42.50</td>
</tr>
<tr>
<td>uscensus</td>
<td>21,782</td>
<td>-0.44</td>
<td>-31.54</td>
<td>-77.68</td>
<td>uscensus</td>
<td>220</td>
<td>-10.45</td>
<td>-20.45</td>
<td>-37.73</td>
</tr>
<tr>
<td>andes</td>
<td>898</td>
<td>-2.23</td>
<td>-27.42</td>
<td>-69.91</td>
<td>andes</td>
<td>336</td>
<td>-0.89</td>
<td>-7.14</td>
<td>-22.92</td>
</tr>
<tr>
<td>hailfinder</td>
<td>46</td>
<td>-5.31</td>
<td>-17.46</td>
<td>-54.71</td>
<td>hailfinder</td>
<td>64</td>
<td>-1.56</td>
<td>+6.25</td>
<td>-15.62</td>
</tr>
<tr>
<td>hepatic2</td>
<td>76</td>
<td>-4.05</td>
<td>-42.56</td>
<td>-70.00</td>
<td>hepatic2</td>
<td>92</td>
<td>-3.26</td>
<td>-21.74</td>
<td>-30.43</td>
</tr>
<tr>
<td>link</td>
<td>7,240</td>
<td>-12.03</td>
<td>-10.58</td>
<td>-61.30</td>
<td>link</td>
<td>1,146</td>
<td>-1.83</td>
<td>-0.44</td>
<td>-22.43</td>
</tr>
<tr>
<td>munion1</td>
<td>497</td>
<td>-7.42</td>
<td>-17.23</td>
<td>-59.14</td>
<td>munion1</td>
<td>208</td>
<td>0.00</td>
<td>+0.96</td>
<td>-9.62</td>
</tr>
<tr>
<td>munion2</td>
<td>7,093</td>
<td>-20.46</td>
<td>-21.66</td>
<td>-43.68</td>
<td>munion2</td>
<td>879</td>
<td>0.00</td>
<td>0.00</td>
<td>-13.31</td>
</tr>
<tr>
<td>munion3</td>
<td>11,558</td>
<td>-36.91</td>
<td>-29.20</td>
<td>-54.19</td>
<td>munion3</td>
<td>898</td>
<td>0.00</td>
<td>0.00</td>
<td>-7.80</td>
</tr>
<tr>
<td>munion4</td>
<td>8,550</td>
<td>-7.87</td>
<td>-13.08</td>
<td>-39.06</td>
<td>munion4</td>
<td>903</td>
<td>0.00</td>
<td>0.00</td>
<td>-8.53</td>
</tr>
<tr>
<td>pathfinder</td>
<td>231</td>
<td>-14.01</td>
<td>-35.38</td>
<td>-69.48</td>
<td>pathfinder</td>
<td>161</td>
<td>-4.35</td>
<td>-8.70</td>
<td>-24.22</td>
</tr>
<tr>
<td>win95pts</td>
<td>132</td>
<td>-6.05</td>
<td>-31.41</td>
<td>-69.07</td>
<td>win95pts</td>
<td>115</td>
<td>0.00</td>
<td>-0.87</td>
<td>-12.17</td>
</tr>
</tbody>
</table>

Table 3: Computation time (seconds). Every result that corresponds to a BDeu score less than 5% smaller than sota-BNSL’s score is boldfaced.

Table 4: Networks’ number of arcs. Every result that corresponds to a BDeu score less than 5% smaller than sota-BNSL’s score is boldfaced.
BN learnt on dataset ‘msnbc’ with sota-BNSL

Figure 1: Bayesian network learnt on dataset ‘msnbc’ with sota-BNSL. BDeu score: -6.19, Nb of arcs: 102, Time until convergence: 252s

how good the tradeoff may be before running any algorithm all the way through, saving us from trying qds-BNSL on datasets in which there are absolutely no strong pairwise relationships, and enabling us to choose an optimal value of $\epsilon$ on datasets for which there is a lot of potential for computational time win with controlled performance loss.

The bound presented in Proposition 4 concerns the MLL score and is far from tight in practice. However, if we could find a tight bound on the BDeu score of the graphs generated by qds-BNSL, it would be much easier to estimate the most promising value of $\epsilon$ for a given dataset.

Finally, we have some insights on ways to generalize our quasi-determinism screening idea. The proof of Proposition 2 suggests that the result still holds when $F$ is any kind of deterministic DAG (and not only a forest). We could therefore use techniques that detect determinism in a broader sense than only pairwise, to make the screening more efficient. For this purpose we could take inspiration from papers of the knowledge discovery in databases (KDD) community, as Huhtala et al. (1999), or more recently Papenbrock et al. (2015) who evaluate functional dependencies discovery methods. We also could broaden our definition of quasi-determinism: instead of considering the information-theoretic quantity $H^D(X|Y)$ to describe the strength of the relationship $Y \rightarrow X$, one could choose $\frac{H^P(X|Y)}{H^P(X)}$, which represents the proportion of $X$’s entropy that is explained by $Y$. Moreover, $\frac{H^P(X|Y)}{H^P(X)} \leq \epsilon$ can be rewritten as $\frac{M^P(X,Y)}{H^P(X)} \geq 1 - \epsilon$, which gives another insight to quasi-determinism screening: for a given variable $X$, this comes down to finding a variable $Y$ such that $M^P(X,Y)$ is high. This is connected to the idea of Chow and Liu (1968), and later Cheng et al. (1997), for whom pairwise empirical mutual information is central. This alternate definition of $\epsilon$—quasi-determinism does not change the algorithms and complexity considerations described in section 4. Lastly, we could consider other definitions of entropy as the ones presented by Rényi et al. (1961).

References


7 Supplementary material

Proof of Lemma 1: First let us rewrite the MLL score in terms of data counts. We denote $x_i [m]$ the $m^{th}$ observation of variable $X_i$ in the dataset $D$. For a given $G \in DAG_V$ and $\theta \in \Theta_G$,

$$l(\theta : D) = \sum_{m=1}^{M} \log(p_0(x_i [m], \ldots, x_n [m])) \prod_{i=1}^{n} \theta_{x_i [m]}^{x_i [m]}$$

$$= \sum_{m=1}^{M} \sum_{i=1}^{n} \log(\theta_{x_i [m]} | x_{\pi(i)} [m])$$

$$= \sum_{i=1}^{n} \sum_{x_i, x_{\pi(i)}} C^D(x_i, x_{\pi(i)}) \log(\theta_{x_i} | x_{\pi(i)})$$

where $C^D(\cdot)$ is the count function associated with $D$: 

$$\forall I \subset V, C^D(x_I) = \sum_{m=1}^{M} \mathbb{I}_{x_I [m]=x_I} = M p^D(x_I).$$

Moreover, it is well known that for categorical variables, the maximum likelihood estimator $\theta^{MLE}$ is given by the local empirical frequencies i.e.

$$\theta^{MLE}_{x_i | x_{\pi(i)}} = p^D(x_i | x_{\pi(i)}) = \frac{C^D(x_i, x_{\pi(i)})}{C^D(x_{\pi(i)})}.$$

Therefore we get:

$$s^L(G : D) = \max_{\theta \in \Theta_G} l(\theta : D)$$

$$= l(\theta^{MLE} : D)$$

$$= \sum_{i=1}^{n} \sum_{x_i, x_{\pi(i)}} C^D(x_i, x_{\pi(i)}) \log(\theta^{MLE}_{x_i} | x_{\pi(i)})$$

$$= \sum_{i=1}^{n} \sum_{x_i, x_{\pi(i)}} M p^D(x_i, x_{\pi(i)}) \log(p^D(x_i | x_{\pi(i)})$$

$$= -M \sum_{i=1}^{n} H^D(X_i | x_{\pi(i)})$$

In other words, $\sigma$ represents an ordering in which each node comes after its parents.

Let $\sigma \in S_n$ be an ordering compatible with $G$. Using the fact that for any variables $X, Y, Z$, $H^D(X|Y) \geq H^D(X|Y, Z)$ we then get that $\forall i \in V \setminus \{\sigma^{-1}(1)\}$,

$$H^D(X_i | x_{\pi(i)}) \geq H^D(X_i | x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(1)})$$

Plugging this inequality in the first equation, reordering the sum according to $\sigma$, and using the chain rule for entropies, we get:

$$-\frac{s^L(G : D)}{M} \geq \sum_{i=1}^{n} H^D(X_i | x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(1)})$$

$$= H^D(X_{\sigma^{-1}(1)})$$

$$+ \sum_{\sigma(i)=2}^{n} H^D(X_{\sigma^{-1}(1)}, \ldots, X_{\sigma^{-1}(1)})$$

Therefore, $\sigma$ is upper-bounded by the $H^D(X_{\sigma^{-1}(1)}), \ldots, x_{\sigma^{-1}(1)})$.

Proof of Proposition 1: Let $G \in DAG_V$ with $V = \llbracket 1, n \rrbracket$ and $D$ containing observations of $X_1, \ldots, X_n$ respectively associated with nodes in $V$.

First, we notice that $s^L(G : D)$ is upper-bounded by the score of a dense DAG. We have shown in Lemma 1 that:

$$s^L(G : D) = -M \sum_{i=1}^{n} H^D(X_i | x_{\pi(i)})$$

It is commonly known that all DAGs are compatible with at least one ordering of the nodes, i.e. that $\exists \sigma \in S_n$ such that

$$\forall i, j \in V \text{ s.t. } j \in \pi^G(i), \sigma(j) < \sigma(i).$$

Proof of Proposition 2: Let $F = \bigcup_{k=1}^{p} T_k$ and $G^*_{R(F)}$ be as in the Proposition’s hypotheses. Without loss of generality, we consider $i$ to be the root of the tree $T_i$. Therefore, $R(F) = \llbracket 1, p \rrbracket$.

Let us also define the following root function that associates to each node the root of the tree it belongs to:

$$r : \begin{align*}
V & \rightarrow R(F) \\
i & \mapsto k \text{ s.t. } X_i \in V_{T_k}.
\end{align*}$$
Let \( G^*_{R(F)} \in \text{DAG}_{R(F)} \) such that:
\[
G^*_{R(F)} \in \arg\max_{G \in \text{DAG}_{R(F)}} s^L(G : D)
\]
and \( G^* = F \cup G^*_{R(F)} \) i.e.
- \( V_{G^*} = V \)
- \( A_{G^*} = (\bigcup_{k=1}^p A_{T_k}) \cup A_{G^*_{R(F)}} \)

We will show as in the proof of Proposition 1 that
\[
s^L_{\text{DAG}}(G^* : D) \geq \max_{G \in \text{DAG}_{V}} s^L_{\text{DAG}}(G : D)
\]
which implies that \( G^* \in \arg\max_{G \in \text{DAG}_{V}} s^L_{\text{DAG}}(G : D) \).

We write:
\[
s^L_{\text{DAG}}(G^*) = -M \sum_{i=1}^n H^D(X_i | X_{\pi^G(i)})
= -M \sum_{i=1}^p H^D(X_i | X_{\pi^G(i)})
\]
\[
= \sum_{i=1}^n H^D(X_i | X_{\pi^{G^*}(i)})
\]

We then compute separately the terms (a) and (b):

- **Computation of (a)**
  The first term corresponds to the score of the graph \( G^*_{R(F)} \) as an element of \( \text{DAG}_{R(F)} \).
  Indeed, by construction of \( G^*_{R(F)} \),
  \[
  \forall i \in R(F), \pi^{G^*}(i) = \pi^{G^*_{R(F)}}(i).
  \]
  Moreover, \( G^*_{R(F)} \) maximizes the MLL score on \( \text{DAG}_{R(F)} \).
  We can now write:
  \[
  (a) = -M \sum_{i=1}^p H^D(X_i | X_{\pi^{G^*_{R(F)}}(i)})
  \]
  \[
  = s^L(G^*_{R(F)} : D)
  \]
  \[
  = \max_{G \in \text{DAG}_{R(F)}} s^L(G : D_{R(F)})
  \]
  \[
  = -MH^D(X_1, \ldots, X_p).
  \]

- **Computation of (b)**
  By construction of \( G^* \),
  \[
  \forall i \in V \setminus R(F), \pi^{G^*}(i) = \pi^{T^*_i}(i).
  \]
  Moreover since the \( T_k's \) are deterministic trees, it follows that
  \[
  \forall i \in V \setminus R(F), H^D(X_i | X_{\pi^{T^*_i}(i)}) = 0.
  \]
  Therefore we can write
  \[
  (b) = -M \sum_{i=p+1}^n H^D(X_i | X_{\pi^{G^*}(i)})
  \]
  \[
  = -M \sum_{i=p+1}^n H^D(X_i | X_{\pi^{T^*_i}(i)})
  \]
  \[
  = 0.
  \]

Collecting the above results yields
\[
s^L_{\text{DAG}}(G^*) = (a) + (b)
= -MH^D(X_1, \ldots, X_p)
\geq -MH^D(X_1, \ldots, X_n)
= \max_{G \in \text{DAG}_{V}} s^L(G : D).
\]

In a few words: the idea of the proof relies on the fact that \( H^D(X) = H^D(X_{R(F)}) \): the information associated with all the variables \( X_1, \ldots, X_n \) is entirely contained in the root variables \( X_{R(F)} \).

**Proof of Proposition 3:** Let \( D \) and \( \epsilon \) be objects that satisfy the input constraints of Algorithm 1, and let \( F_\epsilon \) be the object that is returned by Algorithm 1 with inputs \( D \) and \( \epsilon \).

\( F_\epsilon \) is a directed graph, by definition. Moreover, it is built so that all of its nodes had at most one parent (line 12).

To conclude, we therefore only have to prove that \( F_\epsilon \) does not contain cycles.

Let us suppose that there is a cycle \( i_1, \ldots, i_p \) in \( F_\epsilon \). There are two cases:

1. Either all associated variables have the same entropy: \( H^D(X_{i_1}) = H^D(X_{i_2}) = \cdots = H^D(X_{i_p}) \).
   In which case, there is necessarily two successive nodes in the cycle \( i_l, i_{l+1} \) such that \( i_l < i_{l+1} \).
   However, \( H^D(X_{i_l} | X_{i_{l+1}}) = H^D(X_{i_{l+1}} | X_{i_l}) \leq \epsilon \),
   which means that when the algorithm reaches line 4, \( i_{l+1} \in \pi_\epsilon(i_l) \) and \( i_l \in \pi_\epsilon(i_{l+1}) \).
   Since \( i_l \) is treated before \( i_{l+1} \) in the for loop, this would result in \( i_l \) being removed from \( \pi_\epsilon(i_{l+1}) \),
   thus preventing for \( i_l \) to ever be \( i_{l+1} \)'s parent: we have a **contradiction**.

2. Either there exist at least two variables in the cycle that do not have the same entropy: \( H^D(X_{i_k}) \neq H^D(X_{i_{k'}}), \) for \( k, k' \in \{1, p\} \).
In this case, there also exist two successive nodes $i_l, i_{l+1}$ such that $H^D(X_{i_l}) \neq H^D(X_{i_{l+1}})$.
Let us suppose that $H^D(X_{i_{l+1}}) > H^D(X_{i_l})$. In that case:

$$
H^D(X_{i_l}|X_{i_{l+1}}) = H^D(X_{i_{l+1}}|X_{i_l})
+ H^D(X_{i_l}) - H^D(X_{i_{l+1}})
< 0
< H^D(X_{i_{l+1}}|X_{i_l})
\leq \epsilon.
$$

Therefore when the algorithm reaches line 4, $i_{l+1} \in \pi_{\epsilon}(i_l)$. But when treating either $i_l$ or $i_{l+1}$ during the for loop of lines 4-9, the test on line 6 necessarily implies that $i_l$ is removed from $\pi_{\epsilon}(i_{l+1})$ (since $H^D(X_{i_l}|X_{i_{l+1}}) < H^D(X_{i_{l+1}}|X_{i_l})$). This is in contradiction with the fact that $i_l$ is $i_{l+1}$’s parent.

Therefore, $H^D(X_{i_{l+1}}) < H^D(X_{i_l})$, and arcs in $A_F$ follow nonincreasing entropies, with at least one decrease since we suppose that all entropies are not equal. This is not possible in a cycle: there is a contradiction.

We conclude that there is no cycle in $F_\epsilon$. ■

**Proof of Proposition 4:** The structure of the proof is the same as the one from Proposition 2. The only difference lies in the computation of term $(b)$:

$$(b) = -M \sum_{i=p+1}^{n} H^D(X_i|X_{\pi^{G^*}(i)})
= -M \sum_{i=p+1}^{n} H^D(X_i|X_{\pi^{T_{\epsilon}(i)}})
\leq -M(n-p)\epsilon
\geq -Mn\epsilon.
$$

plugging this in the separated expression of the MLL score of $G^*$ in terms $(a)$ and $(b)$ yields the wanted result. ■