Bayesian networks for static and temporal data fusion

Thibaud Rahier

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques Appliquées

Arrêté ministériel : 25 mai 2016

Présentée par

Thibaud RAHIER

Thèse dirigée par **Florence FORBES**, DR, INRIA
et codirigée par **Stéphane GIRARD**, Directeur de recherche, Inria
préparée au sein du **Laboratoire Institut National de Recherche en Informatique et en Automatique**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Réseaux Bayesiens pour fusion de données statiques et temporelles

Bayesian networks for static and temporal data fusion

Thèse soutenue publiquement le **11 décembre 2018**,
devant le jury composé de :

Madame Florence FORBES
Directeur de Recherche, INRIA, Directeur de thèse
Monsieur Simon de Givry
Chargé de Recherche, INRA (MIAT), Rapporteur
Monsieur Cassio Polpo de Campos
Associate Professor, Universiteit Utrecht, Rapporteur
Monsieur Tomi Silander
Senior Scientist, Naver Labs Europe, Examinateur
Monsieur Philippe Leray
Professeur, Université de Nantes, Examinateur
Monsieur Stéphane Lecoeuche
Professeur, Ecole Mines-Télécom, Examinateur
Monsieur Stéphane Girard
Directeur de Recherche, INRIA, Co-directeur de thèse
Monsieur Sylvain Marié
Ingénieur, Schneider Electric, Examinateur

Abstract

Prediction and inference on temporal data is very frequently performed using time series data alone. We believe that these tasks could benefit from leveraging the contextual metadata associated to time series - such as location, type, etc. Conversely, tasks involving prediction and inference on metadata could benefit from information held within time series. However, there exists no standard way of jointly modeling both time series data and descriptive metadata. Moreover, metadata frequently contains highly correlated or redundant information, and may contain errors and missing values.

We first consider the problem of learning the inherent probabilistic graphical structure of metadata as a Bayesian Network. This has two main benefits: (i) once structured as a graphical model, metadata is easier to use in order to improve tasks on temporal data and (ii) the learned model enables inference tasks on metadata alone, such as missing data imputation. However, Bayesian network structure learning is a tremendous mathematical challenge, that involves a NP-Hard optimization problem. We present a tailor-made structure learning algorithm, inspired from novel theoretical results, that exploits (quasi)-determinist dependencies that are typically present in descriptive metadata. This algorithm is tested on numerous benchmark datasets and some industrial metadatasets containing deterministic relationships. In both cases it proved to be significantly faster than state of the art, and even found more performant structures on industrial data. Moreover, learned Bayesian networks are consistently sparser and therefore more readable.

We then focus on designing a model that includes both static (meta)data and dynamic data. Taking inspiration from state of the art probabilistic graphical models for temporal data (Dynamic Bayesian Networks) and from our previously described approach for metadata modeling, we present a general methodology to jointly model metadata and temporal data as a hybrid static-dynamic Bayesian network. We propose two main algorithms associated to this representation: (i) a learning algorithm, which while being optimized for industrial data, still generalizes to any task of static and dynamic data fusion, and (ii) an inference algorithm, enabling both usual tasks on temporal or static data alone, and tasks using the two types of data.

Finally, we discuss some of the notions introduced during the thesis, including ways to measure the generalization performance of a Bayesian network by a score inspired from the cross-validation procedure from supervised machine learning. We also propose various extensions to the algorithms and theoretical results presented in the previous chapters, and formulate some research perspectives.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Sets

$\mathbb{R}$	Set of real numbers
$\mathbb{N}$	Set of nonnegative integers
$\mathbb{Z}$	Set of integers
$[\![1, n]\!]$	Set of integers $1, \dots, n$
$\mathcal{S}_n$	Set of all permutations of $[\![1, n]\!]$
$\mathcal{F}(U, V)$	Set of functions defined on set U and having values in V
$\mathcal{F}^P(U)$	Set of probability distributions defined on set U
2^V	Power set (set of all subsets) of set V
$\in$	'in' symbol
$\times$	Cartesian product symbol
$\setminus$	Set difference symbol

Random variables and probability distributions

X	Random variable, Static random variable
$Val(X)$	Set of values of the random variable X
$\mathbf{X}$	Tuple of n random variables
D	Dataset containing M rows and n columns
$P_{\mathbf{X}}, P(\mathbf{X})$	Probability distribution of $\mathbf{X}$
P_Θ	Distribution parametrized by Θ
p_X	Distribution function of a categorical variable X
f_X	Density function of a continuous variable X
p_Θ	Distribution function parametrized by Θ
f_Θ	Density function parametrized by Θ
i.i.d	'independent and identically distributed' abbreviation
$P^{\otimes M}$	Distribution of M *i.i.d* variables with distribution P
$t_1, \dots, t_l$	Sequence of successive time stamps
$\mathbf{X}(t)$	Tuple of temporal random variables considered at time t
$\{\mathbf{X}(t_i)\}_{1 \leq i \leq l}$	Time series associated with stochastic process $\mathbf{X}$
$H(X)$	Shannon entropy of a random variable X

$\mathcal{H}(p)$	Shannon entropy of a distribution p
$\mathcal{H}(p \mid\mid q)$	Cross-entropy of distribution q with respect to p
$D_{KL}(p \mid\mid q)$	Kullback-Leibler divergence from distribution q to p
$\mathcal{L}_D(\Theta)$	Likelihood of parameter set Θ given dataset D
$l_D(\Theta)$	Log-likelihood of parameter set Θ given set D

Graphs and Bayesian networks

V	Set of nodes of a graph
I, J, E, Q	Subsets of V
A	Set of arcs of a graph
$G = (V, A)$	Directed acyclic graph (DAG)
$\mathcal{R}(G)$	Roots of DAG G
$\pi^G(i)$	Parents set of node i in DAG G
$Ansc^G(i)$	Ancestors of node i in DAG G
$Desc^G(i)$	Descendants of node i in DAG G
Θ	Set of parameters defining a global distribution of $\mathbf{X}$
$\mathcal{B} = (G, \Theta)$	Bayesian network with structure G encoding distribution P_Θ
$\mathcal{P}(G)$	Number of free parameters associated with the structure G
Θ_i	Set of parameters defining the local distribution of X_i given $\mathbf{X}_{\pi^G(i)}$
ϑ_G	Set of parameter sets Θ such that (G, Θ) is a Bayesian network
$\boldsymbol{\beta}$	Set of coefficients
$\mathbf{x}\boldsymbol{\beta}^T$	Scalar product of vectors $\mathbf{x}$ and $\boldsymbol{\beta}$

HSDBN specific notations

$\mathbf{X}$	Tuple of static variables
$\mathbf{Y}(t_j)$	Tuple of temporal variables considered at time t_j
$\tilde{\mathbf{Y}}(t_j)$	Tuple of temporal variables corresponding to $\mathbf{Y}(t_{j+1})$
$D^{\mathbf{X}}$, $D^{\mathbf{XY}}$	Datasets containing observations of $\mathbf{X}$, $(\mathbf{X}, \mathbf{Y})$ respectively

Other notations

$\mathbb{I}_.$	Indicator function
$\cup, \sqcup$	Union symbol, disjoint union symbol
$\exists$	'there exist' symbol
$\forall$	'for all' symbol
$\subset$	'is a subset of' symbol
iff	'if and only if' abbreviation
w.r.t.	'with respect to' abbreviation

$\alpha_{\mathbf{x}}$	Quantity depending on the value $\mathbf{x}$ of $\mathbf{X}$
$\{\alpha_{\mathbf{x}}\}_{\mathbf{x}}$	Set of all $\alpha_{\mathbf{x}}$ for $\mathbf{x}$ in $Val(\mathbf{X})$
$\sum_{\mathbf{x}} \alpha_{\mathbf{x}}$	Sum of all $\alpha_{\mathbf{x}}$ for $\mathbf{x}$ in $Val(\mathbf{X})$
$\mathbb{A}$	Matrix
$\mathbb{A}^T$	Transpose of matrix $\mathbb{A}$
$\mathbb{A}_{\bullet,i}$	Column i of matrix $\mathbb{A}$
$\mathbb{A}_{i\bullet}$	Row i of matrix $\mathbb{A}$

Introduction

PhD context

This PhD was pursued under a CIFRE[1] agreement between the Inria laboratory and Schneider Electric's Analytics and Artificial Intelligence (AAI) team.
With the increase of the amount of data and sensors that collect data, Schneider Electric has taken a growing interest in machine learning in the last decades, with many successful applications ranging from energy optimization in buildings to fault detection or asset maintenance.

Temporal and static data

Data is a collection of values, that vary (i) in their nature: they can be categorical, numerical, integers, (ii) in their dimension: they can be simple values, vectors, matrices (images), or higher dimension objects, and more importantly (iii) in their temporality: they can be associated to time stamps and represent information that varies through time, in which case they are **temporal data** (Mamoulis et al., 2009), or be considered as constant in time in which case they are considered as **static data**. Note that the same data can be represented in several ways depending on the considered temporal scale: for example, a video clip can either be considered as a temporal sequence of 2−dimensional objects (images), or as a static 3−dimensional object (the third dimension being time).

In this thesis, we focus on fusion of simple static categorical data and temporal numerical data, which are both typically accessible in a real-world industrial setting. Indeed, data is increasingly collected and generated by software systems whether in social networks (Nguyen et al., 2012), smart buildings (La Tosa et al., 2011; Najmeddine et al., 2012), smart grid (Etherden et al., 2017; Pflaum et al., 2017), industry 4.0 (Desdouits et al., 2016), smart cities, or the IoT in general (Koo et al., 2016; Diaz et al., 2012), and include both:

- temporal data, which corresponds to observations of quantities (temperature, energy, amount of carbon dioxide, ...) that evolve in time and are associated with time stamps,

[1] *Conventions Industrielles de Formation par la REcherche*

- static data, which corresponds to what is called **metadata**: a set of descriptive attributes associated with these time series (information concerning the measuring device's location or owner, the measured quantity, ...)

In the context of Schneider Electric, as summarized in Figure 1, there are several types of accessible data that can all be split according to their temporality.

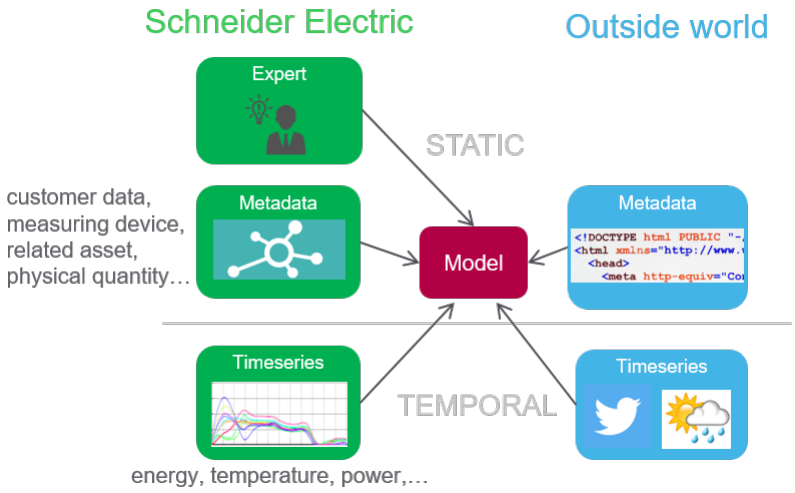


Figure 1: Representation of the different types of data accessible in the context of IoT offers in Schneider Electric, that can be separated into two main categories: **static** and **temporal**

Joint modeling of static and temporal data

Prediction and inference on temporal data is very frequently performed using this type of data alone (Fu, 2011). We believe that these tasks could benefit from leveraging the contextual metadata and that conversely, tasks involving prediction and inference on metadata could benefit from information held within associated temporal data.
Our objective is to design a compact and interpretable model that incorporates both static and temporal data ; to the best of our knowledge there is no standard way to do so.

Jointly modeling temporal data and static metadata would enable:

- to discover *new insights* in databases, for example the fact that machines from a given manufacturer are more likely to have failures,
- to unlock *new applicative capabilities*, such as automatically recovering description associated with sensor data,
- to *automate and improve* tasks that are done today without a joint model of temporal data and static data.

Bayesian networks: a very flexible class of models

In the wide range of accessible Machine Learning models, we are particularly interested in **generative models**, *i.e.* models representing the global distribution of the observed variables, and not only a conditional distribution focusing on a given target variables like **discriminative models**. Generative models enable a wide range of queries without needing to be relearned, naturally handle missing data and diversity in the types of variables. Moreover they are generally interpretable and therefore more suited to be used in an industrial context, in which non-technical experts need to be convinced.
For this reason we focus on Bayesian networks, which are probabilistic graphical models that compactly represent the joint distribution of a set of variables. They have two main purposes:

- They enable **knowledge discovery**, through their structure. This is what we call **interpretability**. For example, they can help to discover interesting conditional independence relations, to identify groups containing variables that are closely related to each other, and understand how variables influence each other.
- They enable **density estimation** thanks to their parameters. They can be interrogated using **inference** to estimate the value of variables given information concerning other variables as evidence.

Outline of the thesis

1. In **Chapter 1**, we first present the PhD subject from a probabilistic point of view: we introduce notations and essential preliminary properties. We then propose a state of the art of Bayesian networks: what they are, how they can be interrogated to perform inference, and how they can be learned from data.

We then present our contributions in the three following chapters of this thesis.

2. In **Chapter 2**, we consider the problem of learning the inherent probabilistic graphical structure of metadata as a Bayesian Network, motivated by the fact that:
   - once structured as a graphical model, metadata is easier to use in order to improve tasks on temporal data,
   - the learned model enables inference tasks on metadata alone, such as missing data imputation.

   However, Bayesian network structure learning is a tremendous mathematical challenge, that involves a NP-Hard optimization problem. We present a tailor-made structure learning algorithm, inspired from novel theoretical results, that exploits (quasi)-deterministic

relations that are typically present in descriptive metadata. This algorithm is then tested on industrial metadatasets and several benchmark datasets. In both cases it proved to be significantly faster than state of the art, with only a limited impact on performance. Moreover, learned Bayesian networks are consistently sparser and therefore more interpretable.

3. Then, in **Chapter 3**, we focus on designing a model that includes both static (meta)data and temporal data. Taking inspiration from state of the art probabilistic graphical models for temporal data (Dynamic Bayesian Networks) and from our previously described approach for metadata modeling, we present a general methodology to jointly model metadata and temporal data as a hybrid static-dynamic Bayesian network. We propose two main algorithms associated with this representation:

    - inference algorithms, targeted towards recovering metadata values from a sequence of temporal data observations,
    - a learning (structure and parameters) algorithm which, while being optimized for industrial data, still generalizes to any task of static and temporal data fusion.

4. Finally, **Chapter 4** is dedicated to a discussion concerning the theoretical and algorithmic results presented in the previous chapters, as well as the presentation of several perspectives.

Chapter 1

Probabilistic framework

Contents

In this chapter, we first present preliminary probabilistic notions and properties, enabling to view temporal and static data as realizations of random variables (Section 1). We then provide an overview of Bayesian networks, directed graphical models that are promising regarding our goal of temporal and static data fusion (Section 2), and expend in particular on Bayesian network learning from observational data (Section 3).

1 Temporal and static data: a probabilistic approach

In this section, we give respective definitions of static and temporal data, and explain their fundamental differences. We then explain what is meant by 'joint modeling' of temporal and static data, along with motivations behind this goal.

1.1 Random variables and datasets

1.1.1 Temporal and static random variables

Let $(\Omega, \mathcal{A}, P)$ be a probability space. A **random variable** X is a function

$$X : \Omega \to Val(X)$$

where $Val(X) = X(\Omega)$ is the image of X.

The set $Val(X)$ can be discrete (finite or infinite) or continuous. For example, a random variable X is said to be:

- **categorical** if $Val(X)$ is finite and unordered,
- **numerical** if $Val(X)$ is a subset of $\mathbb{R}$.

For a given tuple of random variables $\mathbf{X} = (X_1, \ldots, X_n)$, we call a **data point** associated with $\mathbf{X}$ a single realization $\mathbf{x} = (x_1, \ldots, x_n)$ of $\mathbf{X}$, where for $i \in [\![1, n]\!]$, x_i belongs to $Val(X_i)$, the set of possible values of the variable X_i. A collection of data points is referred to as a **dataset**.

Let $\mathbf{x}$ be a given datapoint associated with a tuple of variables $\mathbf{X}$. This data point falls into one of the following two categories.

1. It may be associated with a **time stamp** t. In that case, we consider that the random variable X evolves in time, and that $\mathbf{x}$ is a realization of the **temporal random variable** $\mathbf{X}(t)$, defined as the random variable $\mathbf{X}$ at time t.

2. Such a time stamp may also be unknown, not relevant, or not accessible. In that case we consider that the random variable $\mathbf{X}$ is constant throughout time, and call it a **static random variable**.

A collection of several data points from the first category is an example of **temporal data**, where we observe the evolution of variables through time, whereas a collection of data points from the second category is an example of **static data**, where the information we possess is considered to be constant in time.
These notions are highly dependent on the considered time scale. In this thesis, we consider data as static as long as the characteristic duration needed for its associated variables to change is of a higher order of magnitude than the duration of the experiments during which temporal data is collected, *i.e.* the range of the observation time window.

Example In the context of data coming from connected systems in buildings, we typically have records of temporal data over the span of a few years, whereas descriptive metadata (number of rooms, number of floors, areas of the rooms, number of windows, exposition, ..) usually does not change as long as the building stands, and is therefore considered as static.

1.1.2 Static dataset

A static dataset is a set containing realizations of static random variables. Such datasets are used in an important amount of classical machine learning problems, such as most of those used in online prediction competitions on websites such as `kaggle.com`, `drivendata.org` or `challengedata.ens.fr`.

A static dataset (or simply dataset) associated with a given tuple of variables $\mathbf{X} = (X_1, \dots, X_n)$ is denoted by D.
If D is a dataset containing $M \in \mathbb{N}$ observations of $\mathbf{X} = (X_1, \dots, X_n)$, then for $k \in [\![1, \dots, M]\!]$, $\mathbf{x}^{(k)} = (x_1^{(k)}, \dots, x_n^{(k)})$ is the k^{th} observation of $\mathbf{X}$ recorded in D. In short,

$$D = \{\mathbf{x}^{(k)}\}_{1 \leq k \leq M}.$$

1.1.3 Temporal dataset and time series

Time series Borrowing the definition from Nagarajan et al. (2013), a given sequence of time stamps $\{t_i\}_{i \in I}$ where $I \subset \mathbb{Z}$, a time series is a sequence of successive temporal random variables $\{\mathbf{X}(t_i)\}_{i \in I}$ where for every $i \in I$, $X(t_i)$ corresponds to the random variable $\mathbf{X} = (X_1, \dots, X_n)$ at time t_i.
If $n = 1$, the time series is said to be **univariate**, whereas if $n \geq 2$, it is called **multivariate**.

A **temporal dataset** D is a set containing a time series realization, *i.e.*:

$$D = \{\mathbf{x}(t_i)\}_{i \in I},$$

where for each i, $\mathbf{x}(t_i)$ belongs to the space of accessible values of $\mathbf{X}(t_i)$.
Usually we consider that all the $\mathbf{X}(t_i)$s have the same value space, since they correspond to the same variable at different points in time, *i.e.* $\forall i \in I,\ Val(\mathbf{X}(t_i)) = Val(\mathbf{X})$, even though the distribution of variable $X(t_i)$ depends on the value of i.

In the rest of this thesis, the term **time series** will be used to refer to the sequence of random variables, and **time series dataset** or **temporal dataset** to an associated observation.

1.2 Probability distributions and associated properties

In this section, we present definitions and key preliminary properties that will be used to work with Bayesian networks. We focus on simple random variables, since the extension to tuples is straightforward for all these notions.

1.2.1 Probability distributions

Preliminary: function spaces For any sets U, V, we define $\mathcal{F}(U, V)$ as the set of all possible functions defined on U and having values in V. In particular, if U is finite, $\mathcal{F}_P(U)$ denotes the set of all functions that define a probability distribution on U *i.e.*

$$\mathcal{F}_P(U) = \left\{ p \in \mathcal{F}(U, [0,1]) \mid \sum_{u \in U} p(u) = 1 \right\}.$$

Categorical random variables Let X be a categorical variable. Its probability distribution is denoted by P_X and is entirely defined by its associated function $p_X : \mathcal{F}_P\left(Val(X)\right)$ where:[1]:

$$\forall x \in Val(X),\ P_X(X = x) = p_X(x).$$

Such a distribution can be defined by a parameter $\Theta = \{\theta_x,\ x \in Val(X)\}$, where

$$\forall x \in Val(X),\ \theta_x \in [0,1]\,,$$
$$\sum_{x \in Val(X)} \theta_x = 1.$$

For a given Θ, and for all $x \in Val(X)$, we define:

$$P_\Theta(X = x) = p_\Theta(x) = \theta_x.$$

the associated distribution X parametrized by Θ.

[1]The subscript X may be dropped in practice, both for P_X and p_X.

Continuous random variables: general Let X be a continuous variable. Its distribution is denoted by P_X and is entirely defined by its cumulative distribution function $F_X : \mathbb{R} \to [0, 1]$ where:

$$\forall x \in \mathbb{R}, F_X(x) = P_X(X \leq x).$$

The distribution of a continuous random variable may also be defined by its **density function** [2] $f_X : \mathbb{R} \to \mathbb{R}_+$ such that for all $x \in \mathbb{R}$,

$$F_X(x) = \int_0^x f_X(t)dt.$$

In this thesis, we exclusively model continuous random variables as Gaussian variables.

Gaussian random variable For $\mu, \sigma \in \mathbb{R}_+$ and $\Theta = \{\mu, \sigma\}$, we define the function $f_\Theta(x) : \mathbb{R} \to \mathbb{R}_+$ such that, for all $x \in \mathbb{R}$,

$$f_\Theta(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{(x-\mu)^2}{2\sigma^2}\right).$$

The function f_Θ defines the density function of a **Gaussian random variable** with **mean** μ and **variance** σ^2.
For a continuous variable X with density function f_Θ, we write

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

1.2.2 Conditional probability and associated properties

For simplicity, we focus on categorical variables in this subsection. All properties can be generalized to continuous variables, replacing the probability values by values of the density function. Let $\mathbf{X} = (X_1, X_2)$ a tuple of 2 categorical variables with distribution P.

For all $x_1 \in Val(X_1)$ and $x_2 \in Val(X_2)$, $P(X_1 = x_1 | X_2 = x_2)$ is called the **conditional probability** of $X_1 = x_1$ given that $X_2 = x_2$, and is defined as

$$P(X_1 = x_1 | X_2 = x_2) = \frac{P(X_1 = x_1, X_2 = x_2)}{P(X_2 = x_2)}.$$

The **product rule** is a simple consequence of this definition, and states that, for all $x_1 \in Val(X_1)$ and $x_2 \in Val(X_2)$

$$P(X_1 = x_1, X_2 = x_2) = P(X_1 = x_1 | X_2 = x_2) P(X_2 = x_2).$$

The **Bayes theorem** is a popular rewriting of these equations, and states that for all $x_1 \in Val(X_1)$ and $x_2 \in Val(X_2)$

$$P(X_2 = x_2 | X_1 = x_1) = \frac{P(X_1 = x_1 | X_2 = x_2) P(X_2 = x_2)}{P(X_1 = x_1)}.$$

[2] If such a function exists, which we assume is always the case for the continuous variables encountered in this work.

The **chain rule** for a tuple of n categorical variables $\mathbf{X} = (X_1, \ldots, X_n)$ is an immediate generalization of the product rule. For all $\mathbf{x} = (x_1, \ldots, x_n) \in Val(\mathbf{X})$,

$$\begin{aligned} P(X_1 = x_1, \ldots, X_n = x_n) = P(X_1 = x_1 | X_2 = x_2, \ldots, X_n = x_n) \\ \times P(X_2 = x_2 | X_3 = x_3, \ldots X_n = x_n) \\ \times \ldots \\ \times P(X_n = x_n). \end{aligned}$$

Independence X_1 and X_2 are said **independent**, denoted by $X_1 \perp_P X_2$, if for all $x_1 \in Val(X_1)$ and $x_2 \in Val(X_2)$,

$$P(X_1 = x_1, X_2 = x_2) = P(X_1 = x_1)P(X_2 = x_2).$$

This is equivalent to $P(X_1 = x_1 | X_2 = x_2) = P(X_1 = x_1)$. Intuitively, information concerning the value of X_2 has no influence on our belief of the value of X_1.

Notation simplification For simplification purposes, we may use the notations $P(X_1)$, $P(X_1|X_2)$, $P(X_1, X_2)$ to refer to the distribution of X_1, the conditional distribution of X_1 given X_2 or the joint distribution of X_1 and X_2 respectively.
The definitions stated in this section may be rewritten much more simply using these notations, for example Bayes theorem is simply expressed as:

$$P(X_2|X_1) = \frac{P(X_1|X_2)P(X_2)}{P(X_1)},$$

where it is assumed that the equation holds for all value x_1 and x_2 of variables X_1 and X_2.

1.2.3 Maximum log-likelihood for categorical variables

In this section, we propose a straightforward way to derive the maximum likelihood estimation of the parameters associated with the distribution of a categorical random variable.

Preliminary result This result is a consequence of the Jensen's inequality for the $-\log$ function. A detailed proof is available in Appendix A.1.

Lemma 1 *Let U be a finite set and $p, q \in \mathcal{F}_P(U)$, we have:*

$$\sum_{u \in U} p(u) \log(q(u)) \leq \sum_{u \in U} p(u) \log(p(u)). \tag{1.1}$$

Definitions Let X be a categorical random variable, which distribution is parametrized by Θ. Let $D = \{x^{(m)}\}_{1 \leq m \leq M}$ be a dataset that contains M *i.i.d.* realizations of X.
We define the **count function** C^D as:

$$C^D : \left| \begin{array}{rcl} Val(X) & \longrightarrow & \{0, 1, \ldots, M\} \\ x & \longmapsto & \sum_{m=1}^{M} \mathbb{I}_{\{x^{(m)}=x\}}. \end{array} \right.$$

For $x \in Val(X)$, $C^D(x)$ is the number of times x is observed in the dataset D.
The **likelihood** of Θ given data D is defined as the probability to observe D given that the distribution of X is parametrized by Θ, *i.e.*:

$$\mathcal{L}_D(\Theta) = P_{\Theta}^{\otimes M}(D).$$

where $P_{\Theta}^{\otimes M}$ denotes the distribution of M i.i.d. variables with distribution P_Θ. We denote by $\hat{\Theta}^D$ the **maximum likelihood estimator** (MLE) of Θ relative to D, defined as:

$$\hat{\Theta}^D \in \underset{\Theta \in \mathcal{F}^P(Val(X))}{\operatorname{argmax}} \mathcal{L}_D(\Theta).$$

Under these notations, the following result is well known, and the associated proof is available in Appendix A.1.

Proposition 1 ***Likelihood maximization for a simple categorical variable*** *For X, Θ and D defined as previously, the maximum likelihood estimator of Θ is defined by $\hat{\Theta}^D = \{\hat{\theta}_x^D,\ x \in Val(X)\}$, where for all $x \in Val(X)$:*

$$\hat{\theta}_x^D = \frac{C^D(x)}{M}. \tag{1.2}$$

Remark This result generalizes naturally to a tuple of variables $\mathbf{X} = (X_1, \ldots, X_n)$. In that case, we have $\hat{\Theta}^D = \{\hat{\theta}_{\mathbf{x}}^D,\ \mathbf{x} \in Val(\mathbf{X})\}$ and for any $\mathbf{x} = (x_1, \ldots, x_n) \in Val(\mathbf{X})$:

$$\hat{\theta}_{\mathbf{x}}^D = \frac{C^D(\mathbf{x})}{M},$$

where the count function C^D is naturally extended to $Val(\mathbf{X})$.

2 Bayesian networks: overview

In this section, we provide an overview of Bayesian networks: fundamental definitions and properties, model representation, as well as a presentation of how they can be used for inference tasks. For more exhaustive information, the reader may refer to the very complete books by Koller and Friedman (2009) or Murphy (2012).

2.1 Bayesian networks: representation

2.1.1 Notions of Graph theory

A more exhaustive overview of graphs may be found in Section 1.1 of the book by Nagarajan et al. (2013).

A **graph** G is defined by a tuple (V, E) where

- V is a set of **vertices** or **nodes**, that we will always consider finite in this thesis
- $E \subset V^2$ is a set of distinct **edges**.

Each element $(x, y) \in E$ can be an ordered tuple, in which case it is called a **directed edge** or **arc**, or an unordered tuple, in which case it is called an **undirected edge**. A **directed** (resp. **undirected**) graph is a graph that contains only directed (resp. undirected) edges. In order to prevent any confusion, we choose to denote the set of arcs of a directed graph by A instead of E . The simplest graph structure for a given set of nodes V is the **empty graph**: $G_{empty} = (V, \emptyset)$. The maximally complex structure, in which each node is connected to every other node, is called a **complete graph**. As considered in Nagarajan et al. (2013), graphs representing real-world phenomenon fall in between these two extremes. In the rest of this thesis, a graph $G = (V, E)$ will be considered as **sparse** if the number of edges is of the same order of magnitude as the number of nodes (sometimes denoted by $|E| = \mathcal{O}(|V|)$), and **dense** otherwise.

In a graph $G = (V, E)$, we call a **path** a sequence $v_1, \dots, v_p$ for $p \in \mathbb{N}^*$ such that:

1. $v_1, \dots, v_p \in V$,
2. $\forall i \in \{1, \dots, p-1\},\ (v_i, v_{i+1}) \in E$

If a path $v_1, \dots, v_p$ also verifies $v_1 = v_p$, it is called a **cycle**.
If a path is composed of only directed edges (or arcs), we say it is a **directed path**. We defined analogously a **directed cycle**.

A **directed acyclic graph** (DAG) is a directed graph that *does not contain any directed cycle*. The structure of a DAG $G = (V, A)$ induces a partial ordering $\prec$ of the nodes, which satisfies:

$$\forall v_i, v_j \in V,\ v_i \prec v_j \Rightarrow (v_j, v_i) \notin A.$$

For a DAG $G = (V, A)$, and $V = \{v_i\}_{1 \leq i \leq n}$, we say that an ordering $\sigma \in \mathcal{S}_n$ is a **topological ordering** relative to G if and only if:

$$\forall i, j \in [\![1, n]\!],\ \sigma(i) < \sigma(j) \Rightarrow (v_j, v_i) \notin A.$$

we will indistinctly say that σ is **consistent** with G.

For a DAG $G = (V, A)$, and $v_1, v_2, v_3 \in V$, we say that (v_1, v_2, v_3) form a **V-structure** if $(v_1, v_2) \in A$ and $(v_3, v_2) \in A$.

For a DAG $G = (V, A)$, we define the following **parent** function:

$$\pi^G : \left| \begin{array}{ccc} V & \longrightarrow & 2^V \\ v & \longmapsto & \{w \in V \mid (w, v) \in A\}. \end{array} \right.$$

the exponent G will be dropped for clarity in the rest of this thesis when the referred graph is obvious from context.
The set of **ancestors** of a node v in G is defined as all the nodes from which there is a directed path to v in G, or more formally using the parent function:

$$Anc^G(v) = \{y \in V \mid \exists k \geq 1 \ s.t. \ y \in (\pi^G)^k(\{v\})\},$$

where $(\pi^G)^k$ denotes k successive applications of the parent function, which is canonically extended to sets of nodes.
We can define in a symmetrical way **children** (Ch^G) and **descendants** ($Desc^G$) of a node in a DAG.

The set of **roots** of a DAG G is the set of nodes $v \in V$ for which $\pi^G(v) = \emptyset$. By definition of a DAG, G has at least one root and we note:

$$\mathcal{R}(G) = \{v \in V \mid \pi^G(v) = \emptyset\}.$$

A DAG T is called a **tree** if all of its nodes but one have exactly one parent. For a tree T, we have $|\mathcal{R}(T)| = 1$.
More generally, we call a **forest** a DAG which connected components are trees, *i.e.* a DAG for which each node has **at most** one parent.

2.1.2 Bayesian networks: definitions

In this thesis, we will focus on **static** Bayesian networks modeling only **categorical** variables, and **dynamic** Bayesian networks modeling only **continuous** variables, as this corresponds to the most common respective contexts for the studies of these models, and it is consistent with the real-world data on which this PhD research was based.
In this subsection, however, we still give the main insights on how to use Bayesian networks to model categorical, continuous and mixed variables, as this will be needed for the definition of the model proposed in Chapter 3. Moreover, all modeling details explained in the case of static Bayesian networks (notably concerning the network parameters) can naturally be applied in the context of dynamic Bayesian networks as well, thus preventing the hassle of re-explaining basic definitions in the next subsection.

Setting Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a tuple of categorical random variables.
The distribution of $\mathbf{X}$ is denoted by, $\forall\ \mathbf{x} = (x_1, \ldots, x_n) \in Val(\mathbf{X})$,

$$p_{\mathbf{X}}(\mathbf{x}) = P_{\mathbf{X}}(X_1 = x_1, \ldots, X_n = x_n).$$

For $I \subset [\![1, n]\!]$, we define $\mathbf{X}_I = (X_i)_{i \in I}$, and the previous notations are naturally extended. For example:[3]

$$\forall I, J \subset [\![1, n]\!],\ \forall (\mathbf{x}_I, \mathbf{x}_J) \in Val(\mathbf{X}_I \times \mathbf{X}_J),$$
$$p_{\mathbf{X}_I|\mathbf{X}_J}(\mathbf{x}_I|\mathbf{x}_J) = P_{\mathbf{X}_I|\mathbf{X}_J}(\mathbf{X}_I = \mathbf{x}_I|\mathbf{X}_J = \mathbf{x}_J).$$

Analogous notations are used for continuous variables, where density function f replaces probability function p in all of the equations.

Moreover, we suppose D is a dataset containing M *i.i.d.* realizations of $\mathbf{X}$. All quantities empirically computed from D are written with a $.^D$ exponent.
For instance, p^D refers to the **empirical distribution** with respect to D, defined as:

$$\forall \mathbf{x} \in Val(\mathbf{X}),\ p^D(\mathbf{x}) = \frac{C^D(\mathbf{x})}{M}.$$

Finally, D_I refers to the restriction of D to the observations of $\mathbf{X}_I$.

A **Bayesian network** (BN), first introduced by Pearl (1988), is an object

$$\mathcal{B} = (G, \Theta)$$

where:

- $G = (V, A)$ is a DAG with V the set of nodes and A the set of arcs.
  We suppose $V = [\![1, n]\!]$ where each node $i \in V$ is associated with the random variable X_i
- $\Theta = \{\Theta_i\}_{i \in V}$ is a set of parameters. Each Θ_i defines the local conditional distribution $X_i \mid \mathbf{X}_{\pi(i)}$ of the random variable X_i given the random variables associated to the parent nodes of i, $\mathbf{X}_{\pi(i)}$.
  In the following, we will use the shortcut **parents of** X_i to refer to $\mathbf{X}_{\pi(i)}$ when not ambiguous.

[3] The subscripts $\mathbf{X}_I$ and $\mathbf{X}_J$ will be discarded if no confusion is induced.

Types of variables and modeling choices In the case of mixed Bayesian networks modeling both categorical and continuous variables, categorical variables are generally constrained to have only categorical parent variables, whereas continuous variables can have both categorical and continuous parent variables ((Scutari, 2010; Cobb et al., 2007; Bøttcher et al., 2003; Koller and Friedman, 2009)). Inspired from the modeling choices made in these works, we now detail the three following situations regarding the nature of variables X_i and $\mathbf{X}_{\pi(i)}$ for a given $i \in V$.

- If X_i and $\mathbf{X}_{\pi(i)}$ are categorical variables,

  $$\Theta_i = \{\theta_{x_i|\mathbf{x}_{\pi(i)}}\}_{x_i,\mathbf{x}_{\pi(i)}},$$

  where for $i \in V,\ x_i \in Val(X_i)$ and $\mathbf{x}_{\pi(i)} \in Val(\mathbf{X}_{\pi(i)})$,

  $$\theta_{x_i|\mathbf{x}_{\pi(i)}} = p(x_i|\mathbf{x}_{\pi(i)}).$$

- If X_i and $\mathbf{X}_{\pi(i)}$ are continuous variables, we choose to model them as conditional Gaussian variables. In this case[4],

  $$X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)} \sim \mathcal{N}(\mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T, \sigma_i^2),$$

  and the set of parameters is defined by:

  $$\Theta_i = \{\boldsymbol{\beta}_i, \sigma_i\},$$

  with[5] $\boldsymbol{\beta}_i \in \mathbb{R}^{(|\pi(i)|+1)}$ and $\sigma_i \in \mathbb{R}_+$.
  Formally, for $i \in V,\ x_i \in Val(X_i)$ and $\mathbf{x}_{\pi(i)} \in Val(\mathbf{X}_{\pi(i)})$, we have:

  $$p(x_i|\mathbf{x}_{\pi(i)}) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T)^2}{2\sigma_i^2}\right). \tag{1.3}$$

  In other words, $X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}$ is modeled as a Gaussian variable, with mean $\mu_i = \mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T$ depending linearly on the values $\mathbf{x}_{\pi(i)}$ of $\mathbf{X}_{\pi(i)}$, and with variance σ_i^2.

- if X_i is a continuous variable, and $\mathbf{X}_{\pi(i)}$ is a mix of continuous and categorical variables, we define $\pi_{cat}(i)$ and $\pi_{con}(i)$ such that $\pi(i) = \pi_{cat}(i) \sqcup \pi_{con}(i)$, corresponding respectively to categorical and continuous parent variables of X_i in G.
  In this case, we choose to model the distribution $X_i \mid \mathbf{X}_{\pi(i)}$ as:

  $$X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)} \sim \mathcal{N}(\mathbf{x}_{\pi_{con}(i)}\boldsymbol{\beta}^T_{\mathbf{x}_{\pi_{cat}(i)}}.\sigma^2_{\mathbf{x}_{\pi_{cat}(i)}}).$$

  The associated parameters are defined by:

  $$\Theta_i = \{\boldsymbol{\beta}_{\mathbf{x}_{\pi_{cat}(i)}}, \sigma_{\mathbf{x}_{\pi_{cat}(i)}}\}_{\mathbf{x}_{\pi_{cat}(i)}}.$$

[4] In all this thesis, we use the convention $\mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T$ to denote the scalar product of vectors $\mathbf{x}_{\pi(i)}$ and $\boldsymbol{\beta}_i$.
[5] $\boldsymbol{\beta}_i^T$ has $|\pi(i)|+1$ coefficients because it contains an intercept, which is dropped in the expression $\mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T$ for better readability. This convention does not change any of the derivations, and remains all along this thesis.

In other words, the distribution $X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}$ is modeled as a Gaussian variable, with mean $\mu_i(\mathbf{x}_{\pi(i)})$ which is a linear combination of values of the continuous parent variables $\mathbf{x}_{\pi_{con}(i)}$, with one set of coefficients per configuration of the categorical parent variables. For fixed values of the continuous parent variables $\mathbf{X}_{\pi_{con}(i)}$, X_i is modeled as a mixture of Gaussian variables, with as many components as there are different configurations of its categorical parent variables $\mathbf{X}_{\pi_{cat}(i)}$.

We define ϑ_G as the **parameter space** with respect to G: the set of all possible parameters Θ such that (G, Θ) is a Bayesian network.

For example, in the case where all variables in $\mathbf{X}$ are categorical, ϑ_G is defined as:

$$\vartheta_G = \left\{ \{\Theta_i\}_{1 \leq i \leq n} \mid \forall i, \Theta_i = \{\theta_{x_i|\mathbf{x}_{\pi_{G(i)}}}\}_{x_i, \mathbf{x}_{\pi_{G(i)}}} \right\}$$

where each set of parameters Θ_i is 'legal', *i.e.* $\sum_{x_i \in Val(X_i)} \theta_{x_i|\mathbf{x}_{\pi_{G(i)}}} = 1$, with $\mathbf{x}_{\pi(i)}$ belonging to $Val(\mathbf{X}_{\pi(i)})$.

Example: a simple categorical Bayesian network Let us consider a simple Bayesian network $\mathcal{B} = (G, \Theta)$, modeling two categorical variables X_1 and X_2.

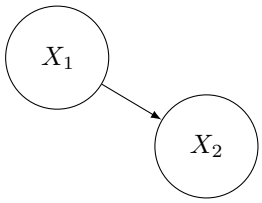


Figure 1.1: Example of a Bayesian network structure

- The structure $G = (\{1, 2\}, \{(1, 2)\})$ is displayed in Figure 1.1.
- The parameters $\Theta = \{\Theta_1, \Theta_2\}$ are defined by:
  - $\Theta_1 = \{\theta_{x_1}\}_{x_1 \in Val(X_1)}$, define the distribution of X_1,
  - $\Theta_2 = \{\theta_{x_2|x_1}\}_{(x_1,x_2) \in Val(X_1) \times Val(X_2)}$ which define the distribution of $X_2 \mid X_1$ (since $\pi^G(2) = \{1\}$)

2.1.3 Bayesian networks and independence

Maps and d-separation Bayesian networks encode a set of conditional independence relations, that correspond to the notion of **d-separation** in a DAG, properly formalized by Verma and Pearl (1988), that we now present.

Definition 1 *Let $G = (V, A)$ be a DAG, and $I, J, C \subset V$ three disjoint subsets of nodes in G. I and J are said to be **d-separated** by C, denoted $I \perp_G J|C$ if along every (undirected) path between a node in I and a node in J, there is a node v, with adjacent nodes in the path v_1 and v_2, satisfying one of the following conditions:*

1. *(v_1, v, v_2) form a V-structure, and none of v or its descendants are in C,*
2. *v is in C and v_1, v, v_2 do not form a V-structure.*

Bayesian networks provide a convenient way to represent probabilistic independence ($\perp_P$) through d-separation ($\perp_G$) induced by the absence of arcs in G. The correspondence between these two notions was formally defined by Pearl (1988) through the notion of **I-map** and **perfect map**:

Definition 2 *Let $G = (V, A)$ be a DAG, and $\mathbf{X}$ a tuple of random variables each of which is represented by a node in V. For all disjoint subsets $I, J, C \subset V$,*

- *G is an independence map or **I-map** of $P_{\mathbf{X}}$ if*

$$I \perp_G J \mid C \Rightarrow I \perp_P J \mid C.$$

- *G is a **perfect map** of $P_{\mathbf{X}}$ if*

$$I \perp_G J \mid C \Leftrightarrow I \perp_P J \mid C.$$

In the case where there exists a perfect map G of $P_{\mathbf{X}}$, not only does $P_{\mathbf{X}}$ factorize in G, but we know that every conditional independence that exists in $P_{\mathbf{X}}$ is encoded in G. In that case we also say that $P_{\mathbf{X}}$ **is faithful to** G.

Probability factorization If $\mathcal{B} = (G, \Theta)$ is a BN representing the distribution of $\mathbf{X}$, then G is an I-map of $P_{\mathbf{X}}$. By a simple application of the chain rule, we get the following factorization of the distribution of $\mathbf{X}$:

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{X}_i | \mathbf{X}_{\pi^G(i)}) \tag{1.4}$$

which holds both for continuous and categorical variables.

For instance, if $X_1, \ldots, X_n$ are all categorical, for all $\mathbf{x} = (x_1, \ldots, x_n) \in Val(\mathbf{X})$ we have:

$$p_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} p_{X_i|\mathbf{X}_{\pi^G(i)}}(x_i|\mathbf{x}_{\pi^G(i)}),$$

and if $X_1, \ldots, X_n$ are all continuous, for all $\mathbf{x} = (x_1, \ldots, x_n) \in Val(\mathbf{X})$, we have:

$$f_{\mathbf{X}}(\mathbf{x}) = \prod_{i=1}^{n} f_{X_i|\mathbf{X}_{\pi_{G(i)}}}(x_i|\mathbf{x}_{\pi_{G(i)}}).^{6}$$

Such a factorization notably implies that *each variable is independent of its non-descendants variables (in G) given its parents variables (in G).*

Markov-equivalence classes Two Bayesian networks $\mathcal{B}_1$ and $\mathcal{B}_2$ are said **Markov-equivalent** when their structures both encode the exact same set of conditional independence relations. The notion of Markov-equivalence being symmetric, reflexive and transitive, each set of Markov-equivalent structures forms an **Markov equivalence class**, and Chickering (1995) shows that the only arcs needed to define a Markov equivalence class are those belonging to at least one V-structure. Equivalence classes are usually represented by completed partially directed acyclic graphs (CPDAGs).

2.1.4 Modeling temporal variables with Bayesian networks

In the same way that we focus on static Bayesian networks modeling only categorical variables, we will focus on dynamic Bayesian networks modeling only continuous variables.

Dynamic Bayesian networks (DBN), first introduced by Dean and Kanazawa (1989), extend the Bayesian network formalism to model dependence relations between temporal variables. Unlike static Bayesian networks that work with multiple *i.i.d.* samples of the variables of interest, each variable modeled with a DBN is represented by several nodes across time stamps, thus enabling the apparition of loops and feedback, which cannot be modeled with static Bayesian networks because of the acyclicity constraint.

A lot of work has been done when it comes to using probabilistic networks to model temporal processes, a vast amount of which is described by Murphy and Russell (2002). We summarize the set of assumptions that are usually made when using dynamic Bayesian networks in the next paragraph, inspired from Nagarajan et al. (2013).

We consider a Bayesian network $\mathcal{B} = (G, \Theta)$ modeling the $n \times T$ variables corresponding to a multivariate time series (or **stochastic process**) $\{(X_1(t), \ldots, X_n(t))\}_{1 \leq t \leq T}$, that has values in $\mathbb{R}^{n \times T}$. The DAG structure G associated with $\mathcal{B}$ has a set of nodes $V = \{v_{it}\}_{1 \leq i \leq n,\ 1 \leq t \leq T}$.
Some very common assumptions concerning the distribution of the considered stochastic process

[6] We sometimes use the notation $f_{X_i}(x_i|\mathbf{X}_{\pi_{G(i)}} = \mathbf{x}_{\pi_{G(i)}})$ instead of $f_{X_i|\mathbf{X}_{\pi_{G(i)}}}(x_i|\mathbf{x}_{\pi_{G(i)}})$ for better readability when the subscripts are too complex, such as in Chapter 3.

are generally made when dealing with dynamic Bayesian networks. These assumptions imply important structural constraints regarding G, that will be summarized hereafter. These constraints enable the creation of a specific dynamic Bayesian network formalism, and of associated learning and inference algorithms.

Assumption 1 *The stochastic process* $\mathbf{X}$ *is first order Markovian: for any* $i, j \in [\![1, n]\!]$ *and* $1 \leq t' < t < T$,

$$X_i(t+1) \perp_P X_j(t') \mid \mathbf{X}(t).$$

In intuitive terms, this assumption means that the variables at time $t+1$ depend only on their immediate past (at time t).

Assumption 2 *For any* $i, j \in [\![1, n]\!]$ *and* $t \in \{1, \ldots, T-1\}$,

$$X_i(t+1) \perp_P X_j(t+1) \mid \mathbf{X}(t).$$

This assumption supposes that time stamps are close enough so that a variable at time $t+1$ is better explained by the immediate past $\mathbf{X}(t)$ than by other variables at time $t+1$.

Assumption 3 *The matrix* $(X_i(t))_{1 \leq i \leq n,\ 1 \leq t \leq T}$ *is full row rank.*

This comes down to assuming that none of the univariate time series $\{X_i(1), \ldots, X_i(T)\}$ for $i \in [\![1, n]\!]$ can be written as a linear combination of the other time series $\{\{X_j(1), \ldots, X_j(T)\}\}_{j \neq i}$.

One last assumption is generally added to enable an easier model representation and estimation:

Assumption 4 *The stochastic process* $\mathbf{X}$ *is* ***homogeneous*** *over time.*

In brief, this supposes that the dependences in between two successive time stamps stay the same during the whole experiment: the structure and associated parameters of the subgraph containing only the variables $(\mathbf{X}(t), \mathbf{X}(t+1))$ is the same for all $t \in \{1, \ldots, T-1\}$.
This assumption is very important when it comes to learning the model parameters: without homogeneity, we would have to learn a different set of parameters (in the form of a $n \times n$ matrix since we are dealing with continuous variables) for the connections between $\mathbf{X}(t)$ and $\mathbf{X}(t+1)$ for each t. This would require a significant number of repeated observations for all variables at each time stamp, which is almost never available in practice.

Structural constraints The three first assumptions imply constraints relative to the graphical structure of the dynamic Bayesian network describing the system (Nagarajan et al., 2013):

- There can be no arcs in between nodes corresponding to variables observed at the same time stamp,
- There can be no arcs in between nodes corresponding to variables observed at nonsuccessive time stamps,
- There can be no arcs 'going back in time'.

Moreover, the fourth assumption enables a last very strong structural constraint:

- All arcs in the dynamic Bayesian network are invariant over time, therefore, the network is entirely defined by its structure in between two consecutive time stamps.

Example of a dynamic Bayesian network Suppose we observe $\mathbf{X} = (X_1, X_2, X_3)$ on regularly spaced time stamps $\{1, 2, \ldots, T\}$, and that $\mathbf{X}$ verifies Assumptions 1, 2 and 3.
Figure 1.2 displays a Bayesian network representing the dependence relations between the set of variables $\{(X_1(t), X_2(t), X_3(t))\}_{1 \leq t \leq T}$.

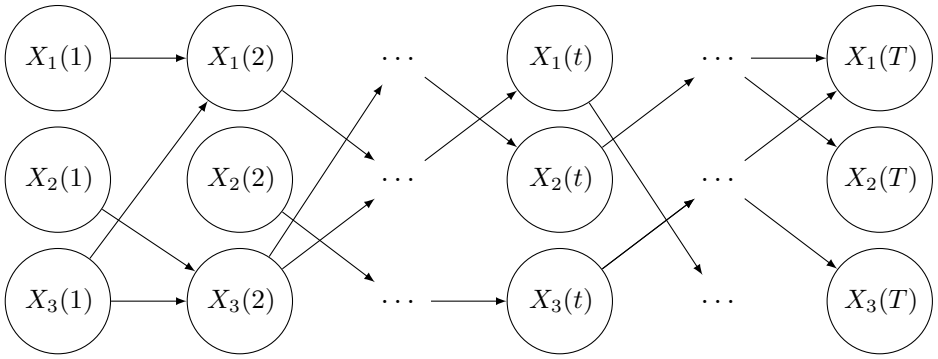


Figure 1.2: Example of a dynamic Bayesian network structure, given that $\mathbf{X} = (X_1, X_2, X_3)$ satisfies Assumptions 1, 2 and 3

If $\mathbf{X}$ also verifies Assumption 4, then the arcs are invariant in time, and the entire network can be represented as a **2-time-slice Bayesian network (2TBN)**, as represented in Figure 1.3.

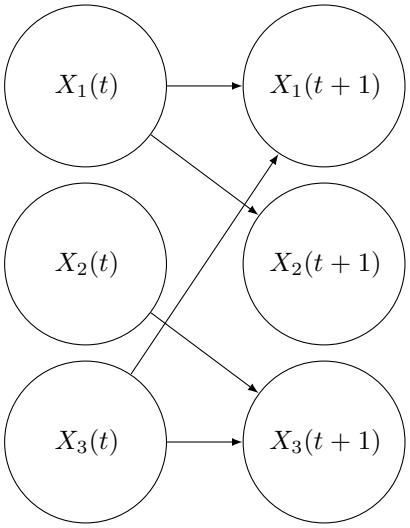


Figure 1.3: Example of a dynamic Bayesian network structure, given that $\mathbf{X} = (X_1, X_2, X_3)$ satisfies Assumptions 1, 2, 3, and 4

2.2 Bayesian networks: inference

In the context of Bayesian networks, questions about the data that go beyond its mere description were introduced by Pearl (1988) as **queries**. The process of using techniques to answer these questions is called **inference**, and was first known as **belief updating**. Recent works such as Koller and Friedman (2009) use this terminology as well.
Inference can be used to make predictions on the state of variables from (partial) observations, but also for *what-ifs* scenarios, where we interrogate the model using virtual observations as evidence.

2.2.1 Reasoning under uncertainty: overview

Posterior estimation Let $\mathcal{B} = (G, \Theta)$ be a Bayesian network modeling the distribution of $\mathbf{X} = (X_1, \ldots, X_n)$, as described in Section 2.1.2, and suppose we have a piece of evidence $\mathbf{E}$. In general, we are interested in studying the effect of $\mathbf{E}$ on the distribution of $\mathbf{X}$, using the knowledge encoded in $\mathcal{B}$. Formally, inference is the process of computing the posterior distribution:[7]

$$P_{\mathcal{B}}(\mathbf{X} \mid \mathbf{E}).$$

Different types of evidence Evidence refers to partial observation or knowledge about a subset $\mathbf{X}_E$ of the variables (with $E \subset [\![1, n]\!]$). There are two main types of evidence:

- **Hard** evidence: an instantiation of $\mathbf{X}_E$. There exists $\mathbf{x}_E^{ev} \in Val(\mathbf{X}_E)$ such that the evidence $\mathbf{E}$ is defined as:

$$\mathbf{E} = \{\mathbf{X}_E = \mathbf{x}_E\}.$$

[7] We will use the $._{\mathcal{B}}$ subscript rather that $._{\Theta}$ since Θ is defined relatively to the structure of $\mathcal{B}$.

- **Soft** evidence: a distribution of $\mathbf{X}_E$ (independent of the way $P(\mathbf{X}_E)$ is modeled by $\mathcal{B}$). Since the structure is fixed in an inference problem, such evidence is specified by a new set of parameters $\{\Theta_e^{\mathbf{E}}\}_{e \in E} \in E$ defining a distribution of the variables $\mathbf{X}_E$. Soft evidence can for example be used to represent unreliable information concerning a subset of the variables.

Different types of queries Typical queries are concerned with a subset of the variables $Q \subset [\![1, n]\!]$. In general, we either want to estimate:

- The **conditional probability** $\mathbf{X}_Q$ given $\mathbf{E}$, *i.e.*

$$P_{\mathcal{B}}(\mathbf{X}_Q \mid \mathbf{E}).$$

  Conditional probability queries (**CPQ**s) have many applications, ranging from hypothesis testing to assessing the odds of a given outcome in different cases of evidence.

- The **maximum a posteriori**: the most probable state of $\mathbf{X}_Q$ given $\mathbf{E}$, *i.e.*

$$\mathbf{x}_Q^* = \underset{\mathbf{x}_Q \in Val(\mathbf{X}_Q)}{\operatorname{argmax}} P_{\mathcal{B}}(\mathbf{X}_Q = \mathbf{x}_Q \mid \mathbf{E}).$$

  MAP queries are often used to impute missing data (using observed data as evidence).

Note that these types of inference can be naturally extended to continuous variables, replacing the probability distribution $P_{\mathcal{B}}$ by the density $f_{\mathcal{B}}$.

2.2.2 Inference algorithms

As seen in the previous subsections, answering queries comes down to estimating posterior probabilities (or their modes). There exist many algorithms for inference, that fall into several categories, summarized in Figure 1.4.

Exact inference Exact inference relies on the use of Bayes theorem and on formal computations to obtain the exact value of the target distribution. Its first general formalization in the context of Bayesian networks is due to Cooper (1990), who also demonstrates its NP-Hardness.
Associated algorithms include: **variable elimination** (Zhang and Poole, 1994; Dechter, 1999), as well as **message passing** (Kim and Pearl, 1983) and **junction trees** (Pearl, 1988), a transformation of the original graph in which the nodes are clustered to narrow the network structure down to a tree, thus guaranteeing speed and convergence of message passing algorithms. Exact inference algorithms are generally used when the Bayesian network structure is reasonably small (typically less than 50 variables), even if they scale to much bigger structures in special cases.

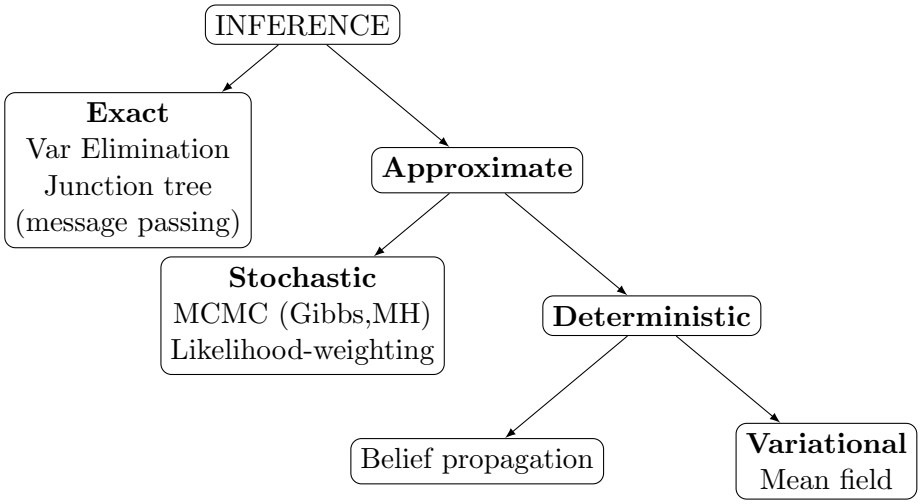


Figure 1.4: Main categories of inference algorithms for Bayesian networks (and some examples)

Approximate inference As we can see in Figure 1.4, approximate inference can be either stochastic or deterministic. These algorithms are more adapted to cases where the number of variables gets greater than 50, as they naturally scale better and are easier to parallelize.

- **Deterministic** approximate inference algorithms include variational methods, that view inference as an optimization problem that can be solved thanks to approximations on the target distribution. The idea was brought to the field of directed graphical models by Saul et al. (1996), and an interesting overview has subsequently been made by Jordan et al. (1999).

  Moreover, usual belief propagation algorithms do not necessarily converge for a general BN structure, and are also used to perform approximate inference.

- **Stochastic** approximate inference algorithms use Monte Carlo simulations from the global joint distribution to estimate the target conditional probability distribution $P_{\mathcal{B}}(\mathbf{X}_Q \mid \mathbf{E})$, or its mode. In the field of computer science, these samples are often referred to as **particles**, and the associated algorithms as **particle filters**. The most commonly known are: **logic sampling** (Henrion, 1988) and its improvement **likelihood-weighting** (Fung and Chang, 1990; Shachter and Peot, 1990). Extensive research has been made on sampling methods, especially on Markov chain Monte Carlo, from the original idea by Metropolis et al. (1953) up to today.

Focus: Likelihood-weighting algorithm In this thesis, we will only face hard evidence and inference will be performed with the likelihood-weighting algorithm (Algorithm 1). It naturally handles evidence $\mathbf{E}$ with small or null $P(\mathbf{E})$, and is therefore able to answer queries that concern both continuous and categorical variables. In consequence, it can be used for inference on both

dynamic and static Bayesian networks, which is a sought-after feature in the context of our research, especially for the models that we introduce in Chapter 3.

We consider that we are given hard evidence $\mathbf{X}_E = \mathbf{x}_E^{ev}$. Moreover, we define $\bar{E} = [\![1, n]\!] \setminus E$. The idea of the Likelihood-weighting algorithm is to constrain every generated sample $\mathbf{x}^{(k)}$ to satisfy the (hard) evidence $\mathbf{E}$, (to set $\mathbf{x}_E^{(k)} = \mathbf{x}_E^{ev}$), then to weight each sample $\mathbf{x}^{(k)}$ by the posterior probability of evidence $\mathbf{E}$ given the values $\mathbf{x}_{\bar{E}}^{(k)}$, *i.e.*

$$w^{(k)} = P_{\mathcal{B}}(\mathbf{X}_E = \mathbf{x}_E^{(k)} | \mathbf{X}_{\bar{E}} = \mathbf{x}_{\bar{E}}^{(k)}).$$

Intuitively, we first sample from a Bayesian network in which the nodes E corresponding to the hard evidence $\mathbf{E}$ are fixed, and then we adjust for the fact that we did not sample from the original Bayesian network by weighting each sample.
This naturally circumvents the problem of very 'rare' evidence, known to cause more naive inference algorithms (such as logic sampling) to demand an astonishing number of samples to return significant results.

The inputs of Algorithm 1 are:

- $\mathcal{B} = (G, \Theta)$: Bayesian network defined on the variables $\mathbf{X} = (X_1, \ldots, X_n)$,
- $E \subset [\![1, n]\!]$: indices of the (hard) evidence variables,
- $\mathbf{Q}$: a given query concerning query variables $\mathbf{X}_Q$ with $Q \subset [\![1, n]\!]$. For simplicity, we focus on a query concerning the probability of a single configuration $\mathbf{x}_Q$ of $\mathbf{X}_Q$, *i.e.* $\mathbf{Q} = \{\mathbf{X}_Q = \mathbf{x}_Q\}$.
- $\mathbf{x}_E^{ev}$: given configuration of the (hard) evidence variables: $\mathbf{E} = \{\mathbf{X}_E = \mathbf{x}_E^{ev}\}$,

$$P_{\mathcal{B}}(\mathbf{Q} \mid \mathbf{X}_E = \mathbf{x}_E^{ev}),$$

- N: number of particles.

This algorithm is programmed into the `bnlearn` `R` package from Scutari (2010), on which the code developed in the context of this thesis is partly based, and has proved to be extremely fast and efficient on the inference tasks that were faced. It is used as a baseline inference algorithm in Section 2 of Chapter 3, where we design new algorithms to perform inference with a new type of Bayesian network, modeling both static and temporal data.

3 Bayesian networks: learning

In the previous section, we have defined Bayesian networks, we have listed some of their interesting properties, and we have seen how they could be used to perform inference. In practice however, one is rarely provided with a Bayesian network when studying a subject, as even experts of the

Algorithm 1 `LikelihoodWeighting`: Inference algorithm

Input: $\mathcal{B} = (G, \Theta)$, E, $\mathbf{x}_E^{ev}$, $\mathbf{Q}$, N

Identify an ordering σ consistent with G, *i.e.* $X_{\sigma(1)} \prec X_{\sigma(2)} \prec \cdots \prec X_{\sigma(n)}$.
Set $w_{\mathbf{E}} = 0$ and $w_{\mathbf{E},\mathbf{Q}} = 0$
for $k = 1$ to N **do**
    for $i = 1$ to n **do**
        if $\sigma(i) \in E$ **then**
            $x_{\sigma(i)}^{(k)} \leftarrow x_{\sigma(i)}^{ev}$
        else
            Generate $x_{\sigma(i)}^{(k)}$ from the distribution of $X_{\sigma(i)} \mid \mathbf{X}_{\pi(\sigma(i))}$ using values $x_{\sigma(j)}^{(k)}$ for $j < i$ (that are defined either by evidence $\mathbf{E}$ or that were previously generated).
    Compute the weight $w^{(k)}$ of the k^{th} sample:

$$w^{(k)} = P_{\mathcal{B}}(\mathbf{X}_E = \mathbf{x}_E^{ev} | \mathbf{X}_{\bar{E}} = \mathbf{x}_{\bar{E}}^{(k)}) = \prod_{e \in E} P_{\mathcal{B}}(X_e = x_e^{ev} | \mathbf{X}_{\pi(e)} = \mathbf{x}_{\pi(e)}^{(k)})$$

    $w_{\mathbf{E}} \leftarrow w_{\mathbf{E}} + w^{(k)}$
    if $\mathbf{x}^{(k)}$ satisfies $\mathbf{Q}$ **then**
        $w_{\mathbf{E},\mathbf{Q}} \leftarrow w_{\mathbf{E},\mathbf{Q}} + w^{(k)}$
$\hat{P}_{\mathcal{B}}(\mathbf{Q} \mid \mathbf{X}_E = \mathbf{x}_E^{ev}) \leftarrow \frac{w_{\mathbf{E},\mathbf{Q}}}{w_{\mathbf{E}}}$
Output: $\hat{P}_{\mathcal{B}}(\mathbf{Q} \mid \mathbf{X}_E = \mathbf{x}_E^{ev})$

associated domain are often not able to draw a precise network structure from their knowledge of the field. It is therefore of great interest to learn Bayesian networks *from observational data*. This is all the more motivated by the fact that our society is increasingly data-centered, and that it is therefore more and more common to have access to important amounts of observations of the variables that we want to model.

After explaining how to solve the parameter learning problem for a Bayesian network with a known structure, we focus on one of the most commonly studied task in the field of Bayesian network learning: *Bayesian network structure learning from observational categorical data*. Several contributions presented in this thesis concern this task, and notably how it can be accelerated in the specific case of IoT metadata and the generalization of this idea to any categorical data (Chapter 2).

We focus on the **complete data case**: we consider the task of learning Bayesian networks from datasets that do not contain any missing value, make the assumption that there are no latent variables.

3.1 Bayesian network parameter learning (known structure)

3.1.1 Maximum likelihood estimation

Setting We consider a Bayesian network $\mathcal{B} = (G, \Theta)$ associated with a tuple of random variables $\mathbf{X} = (X_1, \ldots, X_n)$, for which we know the DAG structure G but ignore the value of the parameters $\Theta \in \vartheta_G$.

Moreover, we suppose we possess a *complete* dataset D containing M *i.i.d.* observations of $\mathbf{X}$. In this context, **parameter learning**, also known as **parameter estimation** can be done in a very straightforward way using maximum likelihood estimation. A very good review of this approach in the context of Bayesian networks has been made by Heckerman (1998).

Log-Likelihood: definition and decomposition We recall the likelihood of the set of parameters $\Theta \in \vartheta_G$ according to the data D is defined as the prob

$$\mathcal{L}_D(\Theta) = P_\Theta^{\otimes M}(D).$$

In practice, we generally consider the log-likelihood:

$$l_D(\Theta) = \log(\mathcal{L}_D(\Theta)).$$

Using the fact that the observations contained in D are independent and identically distributed we can write:

$$\begin{aligned} l_D(\Theta) &= \log(\mathcal{L}_D(\Theta)) \\ &= \log\left(\prod_{m=1}^{M} P_\Theta(\mathbf{X} = \mathbf{x}^{(m)})\right) \\ &= \sum_{m=1}^{M} \log\left(P_\Theta(\mathbf{X} = \mathbf{x}^{(m)})\right) \end{aligned}$$

Injecting the decomposition of the joint probability $P_\Theta(\mathbf{X})$ implied by the Bayesian network structure G yields:

$$\begin{aligned} l_D(\Theta) &= \sum_{m=1}^{M} \log\left(\prod_{i=1}^{n} P_\Theta(X_i = x_i^{(m)} | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}^{(m)})\right) \\ &= \sum_{m=1}^{M} \sum_{i=1}^{n} \log\left(P_{\Theta_i}\left(X_i = x_i^{(m)} | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}^{(m)}\right)\right) \\ &= \sum_{i=1}^{n} \underbrace{\sum_{m=1}^{M} \log\left(P_{\Theta_i}\left(X_i = x_i^{(m)} | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}^{(m)}\right)\right)}_{l_D^i(\Theta_i)}. \end{aligned}$$

Finally, we have:

$$l_D(\Theta) = \sum_{i=1}^{n} l_D^i(\Theta_i), \tag{1.5}$$

where Θ_i is the set of parameters defining the local distribution $X_i \mid \mathbf{X}_{\pi^G(i)}$, and $l_D^i(\Theta_i)$ the associated **local log-likelihood**.
This rewriting of the likelihood is called the **global decomposition** of the likelihood function, and states that the global log-likelihood decomposes in a sum of local log-likelihoods concerning the local conditional distributions $X_i \mid \mathbf{X}_{\pi(i)}$.

Moreover, for $i \in [\![1, n]\!]$, we define

$$G_i = (\{i\} \cup \pi^G(i), \{(j, i),\ j \in \pi(i)\}), \tag{1.6}$$

as the local subgraph of G 'centered' on node i.

Log-likelihood: maximization As in Equation (1), in the setting described in the previous paragraph, the maximum likelihood estimation of the network parameters Θ (MLE, see Section 1.2.3) is defined as follows:

$$\hat{\Theta}^D \in \underset{\Theta \in \vartheta_G}{\operatorname{argmax}}\ l_D(\Theta). \tag{1.7}$$

Proposition 2 stated below was proposed in those terms by Koller and Friedman (2009), relying mainly on ground works by Spiegelhalter and Lauritzen (1990). It states that the global decomposition of the likelihood function given in Equation (1.5) implies an analogous decomposition of the likelihood maximization problem.

Proposition 2 *Let $\hat{\Theta}_i$ be the parameters that maximize the local log-likelihood $l_D^i(\Theta_i)$, i.e.*

$$\forall i \in [\![1, n]\!],\ \hat{\Theta}_i \in \underset{\Theta_i \in \vartheta_{G_i}}{argmax}\ l_D^i(\Theta_i), \tag{1.8}$$

then $\hat{\Theta}^D = \{\hat{\Theta}_i\}_{1 \leq i \leq n}$ satisfies Equation (1.7), i.e. $\hat{\Theta}^D$ is the MLE of the global parameters of the considered Bayesian network

We can therefore maximize the global log-likelihood function $l_D(\Theta)$ by maximizing each local log-likelihood function $l_D^i(\Theta_i)$ independently, enabling to get an efficient solution to the global MLE problem. This is one of the important advantages of modeling a complex joint distribution as a Bayesian network.

3.1.2 Derivation in different cases

Let i be fixed in $[\![1, n]\!]$. We consider the associated subproblem, described in Equation (1.8). In this subsection, we explain how this task is performed, distinguishing between different possible types for the variable X_i and its parent variables $\mathbf{X}_{\pi(i)}$, as discussed in Section 2.1.2:

- X_i can be categorical (case 1), in which case all of its parents are considered to be categorical as well,
- X_i can be continuous (case 2), in which case variables $\mathbf{X}_{\pi(i)}$ can be
  - all continuous,
  - a mix of categorical and continuous variables.

Case 1 - X_i and $\mathbf{X}_{\pi(i)}$ are all categorical In that case, the local log-likelihood $l_D^i(\Theta_i)$ can be rewritten as follows:

$$\begin{aligned} l_D^i(\Theta_i) &= \sum_{m=1}^{M} \log\left(P_{\Theta_i}(X_i = x_i^{(m)} | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}^{(m)})\right) \\ &= \sum_{m=1}^{M} \log\left(\theta_{x_i^{(m)}|\mathbf{x}_{\pi(i)}^{(m)}}\right) / \\ &= \sum_{x_i} \sum_{\mathbf{x}_{\pi(i)}} C^D(x_i, \mathbf{x}_{\pi(i)}) \log\left(\theta_{x_i|\mathbf{x}_{\pi(i)}}\right). \end{aligned}$$

We remind that $C^D(.)$ is the function that counts the number of occurrences of given values in the dataset D, as defined in Section 1.2.3.

Thanks to Proposition 1, we can easily derive the closed form of $\hat{\Theta}_i^D = \{\hat{\theta}^D_{x_i|\mathbf{x}_{\pi(i)}}\}_{x_i, \mathbf{x}_{\pi(i)}}$. For all $x_i \in Val(X_i)$ and $\mathbf{x}_{\pi(i)} \in Val(\mathbf{X}_{\pi(i)})$,

$$\hat{\theta}^D_{x_i|\mathbf{x}_{\pi(i)}} = \frac{C^D(x_i, \mathbf{x}_{\pi(i)})}{C^D(\mathbf{x}_{\pi(i)})}.$$

Note that this corresponds of the empirical probability w.r.t. D, *i.e.* for all $x_i \in Val(X_i)$ and $\mathbf{x}_{\pi(i)} \in Val(\mathbf{X}_{\pi(i)})$,

$$\hat{\theta}^D_{x_i|\mathbf{x}_{\pi(i)}} = p^D(X_i = x_i | \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}).$$

Case 2 - X_i is continuous When the random variable X_i is continuous, each local distribution $X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}$ is modeled as a Gaussian variable, as described in Section 2.1.2.

- In the case where all variables $\mathbf{X}_{\pi(i)}$ are continuous as well, which is generally the setting of Dynamic Bayesian networks, we have seen that the mean $\mu(\mathbf{x}_{\pi(i)})$ is modeled as a linear

function of the values $\mathbf{x}_{\pi(i)}$, and the variance σ_i^2 is supposed independent of $\mathbf{x}_{\pi(i)}$. In that case, the problem comes down to estimating $\boldsymbol{\beta}_i \in \mathbb{R}^{(|\pi(i)|+1)}$ and $\sigma_i \in \mathbb{R}_+$ where:

$$X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)} \sim \mathcal{N}(\mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T, \sigma_i^2),$$

which is equivalent to solving the linear regression problem $X_i = \mathbf{X}_{\pi(i)}\boldsymbol{\beta}_i^T + \varepsilon_i$ where $\varepsilon_i \sim \mathcal{N}(0, \sigma_i^2)$. This problem is simply solvable using standard linear regression formalism.

- In the case where $X_{\pi(i)}$ contains also categorical variables, the mean of $X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}$ is also modeled as a linear regression of its continuous parent values $\mathbf{x}_{\pi_{con}(i)}$, but we have a different coefficient $\boldsymbol{\beta}_i$ of this linear regression *for each configuration* $\mathbf{x}_{\pi_{cat}(i)}$ *of the categorical parents* $\mathbf{X}_{\pi_{cat}(i)}$ *of* X_i.
  The parameter of each linear regression is estimated the same way as in the fully continuous case, with the only subtlety that observations in the data must first be binned according to the different configurations $\mathbf{x}_{\pi_{cat}(i)}$ of $\mathbf{X}_{\pi_{cat}(i)}$.

3.2 Bayesian network structure learning

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

3.2.1 Constraint based algorithms

Constraint-based structure learning algorithms rely on testing for conditional independence relations that hold in the data in order to reconstruct a Bayesian network encoding these independence relations. 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 Yaramakala and Margaritis (2005).

As shown in Section 1 of Chapter 2, constraint-based algorithms are not adapted to the specificities of the data that was used during this research work, namely the presence of very strong pairwise relationships and the fact that some variables have very large sets of configurations. We therefore choose to focus on score and search based structure learning algorithms.

3.2.2 Score and search based algorithms

Background Score and search based structure learning relies on the definition of a network score, then on the search for the best-scoring structure among all possible DAGs. The number of possible DAG structures with n nodes is super-exponential in n, which makes this problem extremely challenging: it has been proven to be NP-Hard by Chickering (1996).

Many score and search based algorithms used in practice 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. Since then, various methods have been proposed: some based on the search for an optimal ordering as in Teyssier and Koller (2005) or Chen et al. (2008), others on the restriction of the structure space by conditional independence testing (sometimes called **hybrid algorithms**), such as the sparse candidate algorithm proposed by Friedman et al. (1999) or the Max-Min Hill Climbing algorithm introduced by Tsamardinos et al. (2006), others on optimizing the search task in accordance to a given score such as Scanagatta et al. (2015)... More recently, several works have focused on optimizing the structure search through theoretical properties arising from the use of given scores or classes of scores, such as de Campos and Ji (2011), or more recently de Campos et al. (2018).

Notations and formalism Suppose we have a *scoring* function $s : DAG_V \to \mathbb{R}$, where DAG_V is the set of all possible DAG structures with node set V. Score-based Bayesian network structure learning comes down to solving the following combinatorial optimization problem:

$$G^* \in \underset{G \in DAG_V}{\operatorname{argmax}} \; s(G). \tag{1.9}$$

We suppose the set V contains n nodes. One can quickly show that:

- $2^{\frac{n(n-1)}{2}}$ is the number of different DAG structures with n nodes that are consistent with a given total ordering,
- $2^{n(n-1)}$ is the total number of directed graphs with no cycle of length one.

This yields:

$$2^{\frac{n(n-1)}{2}} \leq |DAG_V| \leq 2^{n(n-1)}.$$

There are therefore $2^{\mathcal{O}(n^2)}$ possible DAG structures containing n nodes: the size of DAG_V is said to be super-exponential in $|V|$. This gives insight on why Equation (1.9) cannot be consistently solved when n gets typically bigger than 50, even for the most recent algorithms such as those presented by Silander and Myllymäki (2006), Cussens (2011) or Yuan et al. (2013).

Scoring function for Bayesian networks Most scoring functions used in practice are based on the likelihood function introduced previously. The most straightforward being the Max log-likelihood score.
Let $l_D(\Theta) = \log(p_\Theta(D))$ be the log-likelihood of the set of parameters Θ given the dataset D.

For a given DAG structure $G \in DAG_V$, we define the **Max log-likelihood (MLL) score** of G with respect to the dataset D as:

$$s_D^{MLL}(G) = \max_{\Theta \in \vartheta_G} l_D(\Theta). \tag{1.10}$$

The MLL score is very straightforward: it is simply the value of the log-likelihood of the parameters that would be learned (with the MLE approach) given this structure, *i.e.* $s_D^{MLL}(G) = l_D(\hat{\Theta}^D)$. However, it favors denser structures: if $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ are two graph structures such that $A_1 \subset A_2$, we can show that: $s_D^{MLL}(G_1) \leq s_D^{MLL}(G_2)$ (Proposition 3 in Chapter 2). In other words, adding useless arcs to a structure can never decrease the MLL score.

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

- constrain the structure space, for example by bounding the maximum number of parents per node in the final structure,
- use a score that induces a goodness-of-fit versus complexity tradeoff, such as BIC (Schwarz et al., 1978) or BDe (Heckerman et al., 1995).

We define the **Bayesian information criterion (BIC) score** score of $G \in DAG_V$ as follows: (it can vary from a -2 factor, we chose to follow the definition of Koller and Friedman (2009) consistent with the implementation by Scutari (2010)):

$$s_D^{BIC}(G) = s_D^{MLL}(G) - \frac{\log(M)}{2}\mathcal{P}(G)$$

where M is the number of observations in D, and $\mathcal{P}(G)$ is the dimension (*i.e.* number of free parameters) of a Bayesian network with structure G. For example, if the variable $X_1, \ldots, X_n$ are all categorical (the other cases are detailed in Section 3.3.2), we have:

$$\mathcal{P}(G) = \sum_{i=1}^{n} (|Val(X_i)| - 1)\ |Val(\mathbf{X}_{\pi^G(i)})|,$$

where by convention $|Val(\mathbf{X}_\emptyset)| = 1$.

The **Bayesian Dirichlet equivalent (BDe) score** of $G \in DAG_V$ is defined as the log of the marginal likelihood, integrated against a Dirichlet prior[8]. Formally, assuming a uniform prior over all network structures, we have

$$s_D^{BDe}(G) = \log \left(\int_{\Theta \in \vartheta_G} \underbrace{p(D|\Theta, G)}_{\text{Likelihood}} \underbrace{p(\Theta|G)}_{\text{Dirichlet prior}} \, d\Theta \right). \tag{1.11}$$

[8]The Dirichlet prior is chosen in practice mainly because it is conjugate prior for the multinomial distribution: this means that if the prior distribution of the multinomial distribution's parameters is Dirichlet, then the posterior distribution is also Dirichlet. This enables easier interpretation of the prior, and straightforward computations.

In practice, we often use the BDeu score, introduced by Buntine (1991), which is a particular case of BDe with a uninformative Dirichlet prior: all parameters of the Dirichlet prior corresponding to configurations of $X_i \cup X_{\pi(i)}$ for a given i are equal.
In that case, all Dirichlet parameters are proportional to a parameter α, which is called the **equivalent sample size** (ESS). For a given ESS $\alpha > 0$, we define:

$$\forall i \in [\![1, n]\!],\ \alpha_i = \frac{\alpha}{|Val(X_i)| \times |Val(\mathbf{X}_{\pi(i)})|},$$

and the BDeu score is expressed in closed-form as:

$$s_D^{BDe}(G) = \sum_{i=1}^{n} \sum_{\mathbf{x}_{\pi(i)}} \left(\log \left(\frac{\Gamma(\alpha_i |Val(X_i)|)}{\Gamma(C^D(\mathbf{x}_{\pi(i)}) + \alpha_i |Val(X_i)|)} \right) + \sum_{x_i} \log \left(\frac{\Gamma(C^D(x_i, \mathbf{x}_{\pi(i)}) + \alpha_i)}{\Gamma(\alpha_i)} \right) \right)$$

where Γ is the Gamma function:

$$\forall x \in \mathbb{R} \setminus \{0\},\ \Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t} dt.$$

The BDe score is known to be a good indicator of the model's generalization performance, *i.e.* the extent to which the model captures the real underlying distribution. The marginalization of the likelihood (seen in Equation (1.11)) is indeed known to implicitly penalize the number of parameters of the associated model (Rasmussen and Ghahramani, 2001).

In Chapter 2, we will use the BDe score, rather than its asymptotic BIC counterpart (Rusakov and Geiger, 2005), to evaluate Bayesian networks' performance, as it is done in several recent papers such as Vandel et al. (2012) or Nie et al. (2016).

Decomposability and Equivalence All the scores we have mentioned are **equivalent** and **decomposable**.

- An equivalent score assigns the same value to all structures of a given Markov equivalence class.
- A decomposable score is expressed as the sum of local scores (one local score per variable X_i).

The Hill-Climbing algorithm We use an improved version of the hill climbing algorithm, originally presented by Bouckaert (1995) as a baseline for Bayesian network structure learning. Here are the inputs of the `HillClimbing` algorithm, presented in Algorithm 2.

- D: a dataset containing M observations of $\mathbf{X} = (X_1, \ldots, X_n)$,
- $s_D : DAG_n \to \mathbb{R}$ a structure scoring function (depending on the dataset D),
- G: an initial DAG structure with node set $V = [\![1, n]\!]$.

Algorithm 2 `HillClimbing`: Bayesian network structure learning algorithm

Input: D, s_D, G
1: `maxScore` $\leftarrow s_D(G)$
2: `maxScoreIncreases` $\leftarrow$ `TRUE`
3: **while** `maxScoreIncreases` **do**
4: `maxScoreIncreases` $\leftarrow$ `FALSE`
5: **for** all arc addition, deletion or reversal resulting in an acyclic network **do**
6: $G' \leftarrow$ new network
7: `newScore` $\leftarrow s_D(G')$
8: **if** `newScore` $>$ `maxScore` **then**
9: $G \leftarrow G'$
10: `maxScore` $\leftarrow$ `newScore`
11: `maxScoreIncreases` $\leftarrow$ `TRUE`
Output: G

Improvements over the hill-climbing algorithm The hill-climbing algorithm stops when it lands in a local maximum. To avoid cases where this local maximum is far from the global maximum, we use two approaches described notably by Scutari (2010): **tabu list** and **random restarts**.

- A **tabu list** of length `tabuLength` allows the search to decrease the score for `tabuLength` steps, while forbidding the local search to go back on its steps during that time. This enables the algorithm to escape local maxima that are close to better maxima.
- A **random restart** simply consists in running Algorithm 2 from a (new) random initial DAG G. Doing this several times and choosing the best-scoring output structure is a way to avoid bad local maxima.

Algorithm 2 uses three elementary local operators (line 5): arc deletion, addition and reversal. There however has been research on new local move operators, such as 'swapping' proposed by Vandel, Mangin, and De Givry (2012), which combines addition and deletion in order to escape local optima more efficiently.

3.2.3 Going beyond this classification of structure learning algorithms

Among the literature tackling the problem of Bayesian network structure learning, the distinction between constraint-based methods and score and search based methods is almost always clearly stated. However, some works study how these two approaches are in fact linked: for example, Cowell (2001) argues that *for complete data and a given node ordering, the division between constraint-based and score and search based methods is largely a myth* and de Campos (2006) proposes a scoring function for Bayesian network structure learning that is quantitatively linked to the empirical mutual information criterion used in independence tests.

3.3 Dynamic Bayesian network structure learning

3.3.1 Problem specificity

Structural constraints When learning the structure of Dynamic Bayesian networks, we are faced with a particular problem, since we have a set of constraints on the network structure, as summarized in Section 2.1.4.

Structural constraints in general are straightforward to include in most structure learning algorithms (especially iterative heuristics). In practice, it just implies to check if learned arcs satisfy the constraint or not. If not, we simply discard said arc and continue to run the considered algorithm. More generally, any information on the structure that is not coming from the data (knowledge on the problem, intuition...) can be naturally taken into account in structure learning algorithms.

Continuous variables In addition to these structural constraints, dynamic Bayesian networks in this thesis will always be used to model time series corresponding to continuous variables (which is the most common case in the literature as well). In this context, there are two main approaches to learn the structure of a given DBN:

- either we can adapt the structure learning algorithms presented in the case of categorical variables to continuous variables,
- or we can use algorithms that are specific to the learning of probabilistic structures with continuous variables.

3.3.2 Adapting structure learning algorithms to continuous variables

Constraint-based algorithms Adapting constraint based algorithms to continuous variables comes down to testing for independence with continuous variables. This can be done in several standardized ways, such as Pál et al. (2010).

Score-based algorithms Adapting score-based algorithms to continuous variables narrows down to adapt the structure scores to continuous variables. In this thesis, we will focus on the BIC score, which is the most straightforward score to adapt to continuous variables and to mixed categorical and continuous variables. These scores are respectively referred to as the **Gaussian BIC score** and the **Categorical-Gaussian BIC score**.

Suppose we have a Bayesian network $\mathcal{B} = (G, \Theta)$ associated with the tuple $\mathbf{X} = (X_1, \ldots, X_n)$, for which we possess a complete dataset D containing M *i.i.d.* obervations.

The BIC score is decomposable, and can therefore be written as the sum of the local BIC scores:

$$s_D^{BIC}(G) = \sum_{i=1}^{n} s^{BIC}(G_i),$$

where G_i is the local subgraph defined in Equation (1.6).

The local BIC score of $X_i \mid \mathbf{X}_{\pi(i)}$ is called **Gaussian BIC score (bic-g)** if X_i and $\mathbf{X}_{\pi(i)}$ are continuous[9] variables, and **Categorical-Gaussian BIC score (bic-cg)** if X_i is continuous and $\mathbf{X}_{\pi(i)}$ a mix of categorical and continuous variables (corresponding to nodes $\pi_{con}(i)$ and $\pi_{cat}(i)$ respectively). The generic expression of the local BIC score of $X_i \mid \mathbf{X}_{\pi(i)}$ is:

$$s^{BIC}(G_i) = l_D^i(\Theta_i) - \frac{\log(M)}{2}\mathcal{P}(G_i) \tag{1.12}$$

where $l_D^i(\Theta_i)$ is the local maximum log-likelihood, and $\mathcal{P}(G_i)$ is the number of parameters needed to define the conditional distribution $X_i \mid \mathbf{X}_{\pi(i)}$.

Each term of Equation (1.12) depends on the types of variables X_i and $\mathbf{X}_{\pi(i)}$. As previously, three different situations might arise.

- X_i **and** $\mathbf{X}_{\pi(i)}$ **are categorical**. In that case, as we already have seen in Section 3.2.2, Θ_i is the set of parameters $\{\theta_{x_i|\mathbf{x}_{\pi(i)}}\}$ for all $x_i \in Val(X_i)$ and $\mathbf{x}_{\pi(i)} \in Val(\mathbf{X}_{\pi(i)})$, corresponding to values of the local probabilities $P_{\Theta_i}(X_i = x_i|\mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)})$. We can write:

$$\begin{aligned} l_D^i(\hat{\Theta}_i^D) &= \sum_{x_i,\mathbf{x}_{\pi(i)}} C^D(x_i, \mathbf{x}_{\pi(i)}) \log(\hat{\theta}^D_{x_i|\mathbf{x}_{\pi(i)}}) \\ &= \sum_{x_i,\mathbf{x}_{\pi(i)}} C^D(x_i, \mathbf{x}_{\pi(i)}) \log\left(\frac{C^D(x_i, \mathbf{x}_{\pi(i)})}{C^D(\mathbf{x}_{\pi(i)})}\right) \\ \mathcal{P}(G_i) &= |Val(\mathbf{X}_{\pi(i)})|(|Val(X_i)| - 1). \end{aligned}$$

- X_i **and** $\mathbf{X}_{\pi(i)}$ **are continuous.** In that case, $\Theta_i = (\boldsymbol{\beta}_i, \sigma_i)$ were $\boldsymbol{\beta}_i$ is a vector of size $|\pi(i)| \times 1$ and σ_i is a scalar. These parameters define the conditional Gaussian distribution $X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}$ through the following relation:

$$X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)} \sim \mathcal{N}(\mathbf{x}_{\pi(i)}\boldsymbol{\beta}_i^T, \sigma_i^2).$$

  Let $f_{\Theta_i}(X_i|\mathbf{X}_{\pi(i)})$ be the density function of this conditional distribution, given in Equation (1.3). The terms of Equation (1.12) are defined as:

$$\begin{aligned} l_D^i(\hat{\Theta}_i^D) &= \sum_{m=1}^{M} \log(f_{\hat{\Theta}_i^D}(x_i^{(m)}|\mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)}^{(m)})) \\ \mathcal{P}(G_i) &= |\pi(i)| + 1. \end{aligned}$$

[9] We remind that all continuous variables are modeled as conditional Gaussian in the context of this thesis.

- X_i **is continuous and** $\mathbf{X}_{\pi(i)}$ **a mix of continuous and categorical**. This is a more general version of the previous one. Let $\pi_{con}(i)$ and $\pi_{cat}(i)$ be the nodes associated with the continuous parent variables and the categorical parent variables of i respectively. In that case, Θ_i is defined as the set of $\theta_{\mathbf{x}_{\pi_{cat}(i)}} = (\boldsymbol{\beta}_{\mathbf{x}_{\pi_{cat}(i)}}, \sigma_{\mathbf{x}_{\pi_{cat}(i)}})$ for all configurations $\mathbf{x}_{\pi_{cat}(i)}$ of $\mathbf{X}_{\pi_{cat}(i)}$. These parameters define the conditional Gaussian distribution through the following relation:

$$X_i \mid \mathbf{X}_{\pi(i)} = \mathbf{x}_{\pi(i)} \sim \mathcal{N}(\mathbf{x}_{\pi_{con}(i)}\boldsymbol{\beta}_{\mathbf{x}_{\pi_{cat}(i)}}, \sigma^2_{\mathbf{x}_{\pi_{cat}(i)}}).$$

Let $f_{\Theta_i}(X_i|\mathbf{X}_{\pi(i)})$ be the density function of this conditional distribution, given in Equation (1.3). The terms of Equation (1.12) are defined as:

$$l^i_D(\hat{\Theta}^D_i) = \sum_{m=1}^{M} \log(f_{\hat{\Theta}^D_i}(x^{(m)}_i|\mathbf{X}_{\pi(i)} = \mathbf{x}^{(m)}_{\pi(i)}))$$
$$\mathcal{P}(G_i) = (|\pi_{con}(i)| + 1)|\pi_{cat}(i)|.$$

3.3.3 Specific algorithms for DBN structure learning

In all the experiments conducted in the context of this thesis, we followed a score and search based approach, and therefore mainly used the **BIC-cg** score presented in the previous subsection, as it naturally enables mixed-type variables handling and is efficiently computable.
However, there exist algorithms that are specific to Bayesian network structure learning with continuous variables. Notably, the partial ordering implied by the structural constraints of dynamic Bayesian networks enables independent search of each variables' parent set, without risking to introduce cycles. For each of the nodes corresponding to variables in the '$t+1$' time stamp, we are looking for the best set of predictors among the variables in the 't' time stamp, *i.e.* in this specific context, Bayesian network structure learning can be seen as a set of variable selection problems.

Formally: thanks to the homogeneity assumption (Assumption 4 in Section 2.1.4), repeated time measurements can be used to perform learning, and notably for linear regression. Suppose we observe repeated measurements of variables $\mathbf{X} = (X_1, \ldots, X_n)$ on time stamps $T = \{t_j,\ j \in J \subset \mathbb{Z}\}$, and that $\forall j \in J,\ t_{j+1} - t_j = \delta$ is constant, then the dynamic Bayesian network modeling the evolution of $\mathbf{X}$ is defined by the following relation:

$$\forall i \in [\![1, n]\!],\ X_i(t_j) \sim \mathcal{N}(\mathbf{X}(t_{j-1})\boldsymbol{\beta}_i^T, \sigma_i^2),$$

where the $\boldsymbol{\beta}_i$s are $n \times 1$ vectors and the σ's are scalars.

The parameters of the DBN are $\mathbf{B} = \{\boldsymbol{\beta}_i\}_{1 \leq i \leq n}$ and $\boldsymbol{\sigma} = \{\sigma_i\}_{1 \leq i \leq n}$, and the structure of the network is given by the nonzeros coefficients of the β_i's. The structure of the DBN is entirely

defined by the nonzero elements of matrix $\mathbf{B}$.

One of the popular approach to tackle variable selection with continuous variables is the **LASSO** method introduced by Tibshirani (1996), which consists in penalizing the usual mean square loss $\sum_{m=1}^{M}\left(x_i^{(m)} - \mathbf{x}_{\pi(i)}^{(m)}\beta_i\right)^2$ by $\lambda||\beta_i||_{L_1}$, *i.e.* the L_1 norm of the β_i's , multiplied by a coefficient λ. Minimizing this penalized loss induces sparsity in the estimated parameter vectors $\hat{\beta}_i$, which is a way to naturally perform variable selection (finding the best λ coefficient is generally done by cross-validation).

3.4 What is really wanted from Bayesian networks?

3.4.1 Causal interpretation: dream

The fact that Bayesian networks are defined by directed acyclic graphical structures raises many false hopes in terms of causal inference. While Bayesian networks are indeed very compact and easy-to-read representations of joint distributions, one must be cautious when it comes to interpreting arrows as causal relationships.

We have seen in Section 2.1.3 that Bayesian networks are only identifiable up to their Markov equivalence class (*i.e.* the set of all Bayesian networks encoding the same set of conditional independence relations), and one can easily see that Bayesian network belonging to the same Markov class can have very different arc orientations.
A simple example is given by a Bayesian network modeling the joint distribution of three variables X, Y and Z: in that case, the structures $X \to Y \to Z$ and $Z \to Y \to X$ encode the same conditional independence $\{Z \perp X|Y\}$, and are therefore in the same equivalence class. However, they contain arcs reversed from each other.

From an intuitive point of view, it could be argued that a Bayesian network should represent the causal structure of the data it is describing. To answer this request, Pearl et al. (2009) introduces **causal Bayesian networks**. However, strong assumptions are needed for these models to be learnable from the data:

- Each variable X_i must be conditionally independent of its non-effect given its direct causes,
- There must exist a network structure which is faithful to the dependence structure of $\mathbf{X}$.

These assumptions notably imply that there is no **latent variable** in the data, *i.e.* no unobserved variable influencing the variables of the network.

In this precise case, such a causal network may be learned from the data, and causal inference may be done. Checking if these assumptions are satisfied is not often doable in practice, and we

therefore do not go further into the theory and use of causal networks. We therefore stick to the less intuitive but more sound interpretation of absence of arcs as conditional independence: *each node is independent of its non-descendants given its parents.*

3.4.2 Generalization performance and readability: reality

We have seen that learning causal relationships is not a reasonable goal, moreover, most algorithms only learn approximately optimal structures (either in terms of score or verified conditional independence), and finally, models are only identifiable up to their Markov equivalence class. The question can then be asked: *what do we really want from Bayesian networks ?*

Going beyond the purely statistical goal of model identification, a reasonable goal with BNs is the two sided 'performance':

- **Qualitative performance** (interpretability / readability). Bayesian networks that are readable are more convincing to non-experts, and can provide insights in a way standard discriminative models cannot.
- **Quantitative performance** (generalization accuracy). Bayesian networks that have a good generalization performance are more accurate when answering queries of any kind, and can therefore perform multiple applicative tasks: prediction, missing data imputation, diagnosis, ...

The validation log-likelihood (VLL) score One possible way to measure how well a generative model captures the underlying distribution of the data is to use the validation log-likelihood (VLL) score. This procedure is notably used for hidden Markov models hyperparameters selection (Celeux and Durand, 2008), but it is rarely considered in the context of Bayesian networks in general.
We place ourselves in the following context: we have M observations of the categorical variables $\mathbf{X} = (X_1, \ldots, X_n)$ in a dataset D. We suppose we have a training set T and a validation set V such that $D = T \sqcup V$.
The VLL score of a DAG G trained on T and validated on V is defined as:

$$s^{VLL}_{T,V}(G) = l_V(\hat{\Theta}^T). \tag{1.13}$$

In other words, it is the log-likelihood of the parameters learned by MLE on the training set T, evaluated on dataset V. This score captures the generalization capability of structure G, as long as this structure was learned independently from the validation V: in practice, G comes either from prior knowledge concerning the variables $\mathbf{X}$, or from a structure learning algorithm that was run on dataset T only.
If T and V are randomly chosen from D, we use the notation $s^{VLL}_D(G)$ to refer to the VLL score

of G, which is in that case a random variable.

This quantity may be estimated by averaging the VLL score for different random partitions (T, V) of D, for example by following the cross-validation procedure, which is a very common approach in supervised learning (Friedman et al., 2001).

If D is split in K random sets $D_1, \ldots, D_K$, and defining, for $k \in [\![1, K]\!]$,

$$T_k = \bigcup_{l \in [\![1,K]\!] \setminus \{k\}} D_l$$
$$V_k = D_k.$$

The K-fold **Cross-validation log likelihood (CVLL) score** of a given structure learning algorithm `algo-BNSL` is defined as[10]:

$$s_D^{CVLL}(\texttt{algo-BNSL}) = \frac{1}{K} \sum_{k=1}^{K} s_{T_k, V_k}^{VLL}(\texttt{algo-BNSL}(T_k)). \tag{1.14}$$

The CVLL score is used to evaluate algorithms rather than structures, since there is no guarantee `algo-BNSL` learns the same structure on the different training sets $\{T_k\}_{1 \leq k \leq K}$.

In Chapter 2, we use both the BDe score and the VLL score (averaged over several random partitions (T, V) of D) to assess the performance of structure learning algorithms.

In Section 1 of Chapter 4, we study the VLL score more in depth, notably exploring how it avoids overfitting in a natural way, and how well it really assesses the generalization ability of a structure learning algorithm.

[10] $s_D^{CVLL}(\texttt{algo-BNSL})$ is a random quantity as well.

Chapter 2

Screening strong pairwise relationships for fast Bayesian network structure learning

Contents

In this chapter, we are interested in finding to what extent determinism, that we define using information-theoretic notions, can be exploited in order to improve Bayesian network structure learning. We focus on structure learning for Bayesian networks that only model categorical variables.
We first define determinism, and explain how Bayesian network structure learning is impacted by its presence in data (Section 1). We then propose several results bridging the gap between the notion of determinism and score&search based structure learning of Bayesian networks (Section 2). These theoretical results are then used to design an algorithm enabling a new approach to Bayesian network structure learning in the presence of determinism (Section 3). This idea is subsequently extended to any kind of data via the notion of quasi-determinism (Section 4). Finally, we present some experiments showing how these algorithms perform in practice, both on reference benchmark datasets and on real-world data from the IoT domain (Section 5). We conclude by discussing these results and stating some perspectives (Section 6).

1 Bayesian network structure learning using data from the IoT domain: a particular problem

In this section, we present two major issues that arise when learning Bayesian networks on internet of things (IoT) descriptive categorical data: the presence of determinism and the fact that some of the variables have a high number of accessible configurations.
First, we formally define the notion of determinism, and show how it relates to functional dependence. We then describe the problems that generally arise when learning Bayesian networks using data that contain determinism, and how past works address this problem. In the second part, we briefly explain how categorical variables with a high number of values are difficult to take into account when learning Bayesian networks with standard methods.

1.1 Determinism

1.1.1 Introduction

Determinism can be found in several types of data, 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 which 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.

1.1.2 Definitions

Determinism is a degenerate case of probabilistic dependency, that is also called **functional**. Formally, if $\mathbf{X}$ is a tuple of random variables and Y a simple random variable, and P a probability

distribution over $(\mathbf{X}, Y)$, the relationship $\mathbf{X} \to Y$ is said to be **functional** or **deterministic** iff there exists a function $f : Val(\mathbf{X}) \to Val(Y)$ such that $\forall (\mathbf{x}, y) \in Val(\mathbf{X}) \times Val(Y)$,

$$P(Y = y|\mathbf{X} = \mathbf{x}) = \mathbb{I}_{\{y=f(\mathbf{x})\}}.$$

i.e. iff for any realization $(\mathbf{x}, y)$ of $(\mathbf{X}, Y)$, we have that $\mathbf{y} = f(\mathbf{x})$.

Remark In the particular case where $\mathbf{X}$ and Y are continuous, and f is linear, this definition coincides with the notion of **co-linearity**.

In this chapter, we focus on the problem of Bayesian network structure learning from categorical data only (without any prior knowledge). In that context, we can only observe **empirical determinism**, that is, with respect to a dataset D.

Definition 3 ***Determinism w.r.t.*** D

Given a dataset D containing observations of $\mathbf{X}_J$ and X_i , the relationship $\mathbf{X}_J \to X_i$ is deterministic with respect to D, iff $H^D(X_i|\mathbf{X}_J) = 0$, where

$$H^D(X_i|\mathbf{X}_J) = -\sum_{x_i, \mathbf{x}_J} p^D(x_i, \mathbf{x}_J) \log(p^D(x_i|\mathbf{x}_J))$$

is the empirical conditional Shannon entropy of X_i given $\mathbf{X}_J$ with respect to D.

We remind that $p^D(x_i|\mathbf{x}_J) = P^D_{X_i|\mathbf{X}_J}(X_i = x_i|\mathbf{X}_J = \mathbf{x}_J)$ is the empirical conditional probability of observing $X_i = x_i$ given that $\mathbf{X}_J = \mathbf{x}_J$, *i.e.*

$$p^D(x_i|\mathbf{x}_J) = \frac{C^D(x_i, \mathbf{x}_J)}{C^D(\mathbf{x}_J)}.$$

It is straightforward to show that:

$$\begin{aligned} &H^D(X_i|\mathbf{X}_J) = 0 \\ &\Leftrightarrow \forall \mathbf{x}_J \in Val(\mathbf{X}_J), \exists! x_i \in Val(X_i) \ s.t. \ p^D(x_i|\mathbf{x}_J) = 1. \end{aligned}$$

Definition 3 is therefore the empirical version of the definition of a functional relationship that we presented: for each possible value $\mathbf{x}_J$ of $\mathbf{X}_J$, there exists a value x_i of X_i such that every time we have an observation $\mathbf{x}^{(m)}$ in D for which $\mathbf{x}^{(m)}_J = \mathbf{x}_J$, we also have $x^{(m)}_i = x_i$.

In the rest of this work, determinism will always implicitly mean empirical determinism with respect to a **given dataset** D. Moreover, we will assume that the sets of variable values $\{Val(X_i)\}_{1 \leq n}$ contain **only values** of the X_i variables **that are observed in** D.

1.1.3 Bayesian network structure learning in presence of determinism

Several research works tackle the problem of Bayesian Network structure learning in presence of deterministic relationships (Scheines et al., 1996; Luo, 2006; Mabrouk et al., 2014; de Morais et al., 2008), but to our knowledge, they focus on constraint-based approaches.
We remind that constraint-based structure learning relies on the identification of conditional independence statements from the data (by running some independence tests) in order to induce DAG structure underlying the joint distribution of the variables. This task assumes that all probabilistic independence relations ($\perp_P$) can be represented by a DAG structure G thanks to the notion of d-separation ($\perp_G$). However, this implication only holds when the underlying distribution of the data P is DAG-faithful, as explained in Chapter 1.

As shown by Luo (2006), the presence of determinism may introduce independence relations that do not correspond to d-separations in the true inherent DAG structure of the data: the faithfulness of the distribution is therefore lost. The following simple example illustrates this phenomenon.

Example of faithfulness failure due to determinism Consider a tuple of categorical random variables $\mathbf{X} = (X_1, X_2, X_3, X_4)$, and an associated DAG structure $G = (\{1,2,3,4\}, \{(1,3),(2,3),(3,4)\})$ represented in Figure 2.1. Let $\mathcal{B} = (G, \theta)$ be a Bayesian network modeling the distribution of $\mathbf{X}$, with $\theta \in \Theta_G$ a set of parameters defining the local conditional distributions $X_i | X_{\pi^G(i)}$.

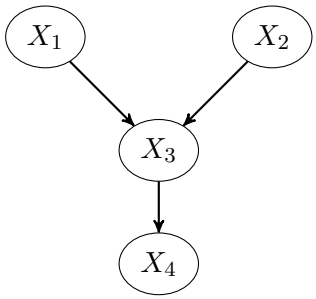


Figure 2.1: Example of Bayesian network structure G

Let P be the distribution encoded by the Bayesian network $\mathcal{B}$, and suppose that the relationship $\mathbf{X}_{\{1,2\}} \to X_3$ is deterministic, *i.e.*

$$H(X_3 | \mathbf{X}_{\{1,2\}}) = 0.$$

Under these assumptions, we can show that P is not DAG-faithful: indeed,

- We have $P(X_3 | \mathbf{X}_{\{1,2\}}, X_4) = P(X_3 | \mathbf{X}_{\{1,2\}})$ since $\mathbf{X}_{\{1,2\}} \to X_3$ is deterministic.
- This implies that $X_3 \perp_P X_4 \mid \mathbf{X}_{\{1,2\}}$ holds.

- However, $X_3 \perp_G X_4 \mid \mathbf{X}_{\{1,2\}}$ does not hold since G contains the arc $X_3 \to X_4$.

By definition (Section 2.1.3 of Chapter 1), P is therefore **not faithful** to G, although G is the true structure underlying P. The distribution P is therefore not DAG-faithful.

More generally, one can show that such a failure of faithfulness consistently happens as soon as a variable that is deterministic given its parent variables has a child variable in the DAG structure underlying the distribution.

The different works that tackle the problem of constraint-based Bayesian network structure learning in presence of deterministic relationships, such as Scheines et al. (1996), de Morais et al. (2008) or Luo (2006), try to circumvent the problem of unfaithfulness, by designing specific algorithms that take special care of deterministic relationships. Mabrouk et al. (2014) argues that, although these methods are efficient, most of them have important shortcomings, such as learning too many or too few arcs.

In this thesis, we choose to take a step back from the statistical problem of model identification which is central in constraint-based methods, and look into the way deterministic relationships (in an empirical sense) influence score and search based Bayesian network structure learning.

1.2 High number of configurations for categorical variables

1.2.1 Description

An important part of descriptive metadata we deal with comes from real-world IoT systems. Some variables in these datasets contain almost unique information for each observation (entry)[1]. These variables have almost as many different values as there are rows in the associated table. As an illustration, out of 47 descriptive attributes and 1000 rows in the `HOMES` metadataset (La Tosa et al., 2011) that we use for the experiments in Section 5.1, 6 attributes have more than 500 values.

This phenomenon is also observed outside of the context of IoT data. For example, data from the `Pump It Up` [2] challenge, that is used in experiments of Section 5.2, contains variables with a high number of values: the dataset has 54k rows, and some of the low-level descriptive variables have more than 10k values.

1.2.2 Memory issues

The number of parameters associated with a Bayesian network modeling one or more of such 'big' variables can be very high.

[1] An extreme example is the described element `id` variable, which has as many values as there are rows.
[2] `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/`

For example, the conditional probability table (CPT) describing the distribution of a variable with 10 values which has three parent variables with 500 values each contains more than 10^{11} parameters. If each parameter was to be stored in a single byte, this CPT would take up more than 100Go of memory space.

This raises issues when searching locally for the best structure in the context of score and search based Bayesian network structure learning, or when running independence tests in the context of constraint based structure learning. Such CPTs must be stored to compute scores or tests statistics, thus massively slowing down the associated structure learning algorithm. This problem is frequently encountered when running benchmark structure learning algorithms (Hill Climbing, MMHC,...) on datasets containing variables with a high number of configurations, and was the initial motivation behind the theoretical and algorithmic contributions presented in the rest of this chapter.

2 Bridging the gap between determinism and the MLL score

In this section, we first remind known results concerning the rewriting of the max log-likelihood (MLL) score for Bayesian networks. We then show several results that lead to Proposition 3, stating that particular cases of tree-structured DAGs can be solutions of the structure learning optimization problem with regards to the MLL score. We finally generalize this result in Proposition 4, and explain the intuition behind the use of this theoretical property to simplify in practice the Bayesian network structure learning task.

In **all this section**, we consider that $\mathbf{X} = (X_1, \ldots, X_n)$ is a tuple of categorical random variables indexed by $V = [\![1, n]\!]$, and that D is a complete dataset containing M observations of $\mathbf{X}$.
The proofs of lemmas and propositions can be found in Appendix A.2.

2.1 Notations and preliminary results

2.1.1 An important lemma

We first recall a lemma that relates the MLL score, presented in Chapter 1, to the notion of empirical conditional entropy. This result is well known and notably stated by Koller and Friedman (2009).

Lemma 2 *For $G \in DAG_V$ associated with variables $X_1, \ldots, X_n$ observed in a dataset D,*

$$s_D^{MLL}(G) = -M \sum_{i=1}^{n} H^D(X_i | \mathbf{X}_{\pi^G(i)})$$

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

2.1.2 The MLL score and complete DAGs

The following lemma formalizes the idea that the MLL score leads to overfitting, by showing it cannot decrease when an arc is added to a DAG structure.

Lemma 3 *Let $G = (V, A)$ be a DAG, and $G' = (V, A \cup \{(i_0, i_1)\})$ a DAG with one more arc $(i_0, i_1) \notin A$. Then, the MLL score difference of these structure is:*

$$s_D^{MLL}(G') - s_D^{MLL}(G) = M\left(I^D(X_{i_0}, X_{i_1} \mid \mathbf{X}_{\pi^G(i_1)})\right).$$

where I^D is the empirical **mutual information**, defined as, for all $I, J, C \subset V$

$$I^D(\mathbf{X}_I, \mathbf{X}_J|\mathbf{X}_C) = \sum_{\mathbf{x}_I, \mathbf{x}_J, \mathbf{x}_C} p^D(\mathbf{x}_I, \mathbf{x}_J, \mathbf{x}_C)\log\left(\frac{p^D(\mathbf{x}_I, \mathbf{x}_J|\mathbf{x}_C)}{p^D(\mathbf{x}_I|\mathbf{x}_C)p^D(\mathbf{x}_J|\mathbf{x}_C)}\right),$$

which has the property of being nonnegative.

Remark For G and G' satisfying the hypothesis of Lemma 3, we have that $s_D^{MLL}(G') \geq s_D^{MLL}(G)$. In other words, **adding an edge to a DAG can only increase its MLL score**.

We are now going to show how this implies that a 'complete' DAG (in the sense that no edge can be added to it without breaking its acyclicity property) maximizes the MLL score among all possible network structures.

In the following definition and lemma, we formally define **complete** DAGs and show that **all complete DAGs have the same MLL score**.

Definition 4 ***Complete DAG***
For $\sigma \in \mathcal{S}_n$ (set of permutations of $[\![1, n]\!]$), the σ-complete DAG is defined as $G^\sigma_{comp} = (V, A^\sigma_{comp})$ where

$$A^\sigma_{comp} = \{(\sigma(i), \sigma(j)) \mid i < j,\ (i, j) \in V^2\}.$$

In other words: G^σ_{comp} contains all the arcs that are consistent with the ordering σ. It is indeed complete because adding any arc would break its acyclicity property.

Lemma 4 *For any $\sigma \in \mathcal{S}_n$,*

$$s_D^{MLL}(G^\sigma_{comp}) = -MH^D(X_1, \ldots, X_n).$$

Remark The MLL score of a complete DAG, does not depend on the ordering σ this DAG is consistent with.

Lemmas 3 and 4 straightforwardly give us the following result:

Lemma 5 *$\forall G \in DAG_V$, the MLL score of G has the following upper bound:*

$$s_D^{MLL}(G) \leq -MH^D(X_1, \ldots, X_n). \tag{2.1}$$

Moreover, the following condition on G is sufficient for this upper bound to be reached:

$$\left(\exists \sigma \in \mathcal{S}_n \mid G = G^{\sigma}_{comp}\right) \Rightarrow \left(s_D^{MLL}(G) = -MH^D(X_1, \ldots, X_n)\right). \tag{2.2}$$

Remark The fact that complete DAGs maximize the MLL score is a consequence of overfitting: the most complex models have the best scores.
A natural question that now comes to mind is: are there other graphs (possibly sparse) that also maximize this score ? If such a graph is found, while also satisfying a complexity constraint (either on the number of parameters, or on the number of edges, number of parents per node, etc), it could be very interesting from a fit-complexity tradeoff standpoint.

In the next subsection we will see examples of such graphs, in the specific case where data contains pairwise determinism.

2.2 Deterministic DAGs and the MLL score

We still consider the setting of a dataset D, containing observations of $\mathbf{X} = (X_1, \ldots, X_n)$.
We now define the notion of deterministic DAG with respect to D.

Definition 5 ***Deterministic DAG w.r.t.*** *D*
$G \in DAG_V$ is said to be deterministic with respect to D iff

$$\forall i \in V \text{ s.t. } \pi^G(i) \neq \emptyset,\ \mathbf{X}_{\pi^G(i)} \to X_i \text{ is deterministic w.r.t. } D.$$

In other words, G is deterministic if every node which has at least one parent in G corresponds to a variable that is entirely determined by its parent variables.

2.2.1 Deterministic trees

The proposition that follows is a natural consequence of the results proven in the previous subsection.

Proposition 3 *If T is a deterministic tree with respect to D, then T maximizes the MLL score among all DAG structures, i.e.*

$$s_D^{MLL}(T) = \max_{G \in DAG_V} s_D^{MLL}(G).$$

Remark A deterministic tree T implies only **pairwise deterministic** relationships, since by definition every node except the root has a single parent in T.

We have seen that complete DAGs also maximize the MLL score. The main interest of Proposition 3 resides in the fact that, under the (strong) assumption that a deterministic tree T exists, it is a sparse solution to the combinatorial optimization problem introduced in Equation (1.9), with $n-1$ arcs (instead of $\frac{n(n-1)}{2}$ for a complete DAG).

Proposition 3 is all the more interesting as the tree T is 'deep'. The existence of such a deep deterministic tree can seem like a very constraining condition. It does however happen naturally in some cases, such as when data is stored in a relational database. In this specific context, we are confident that we can find networks that are sparse, interpretable and which maximize the MLL score.

2.2.2 Deterministic forests

The deterministic tree assumption of Proposition 3 is very restrictive. In this section, we propose an extension of this result to deterministic forests.

First, we show the following straightforward lemma, which states that a deterministic forest can be written as a disjoint union of deterministic trees.

Lemma 6 ***Deterministic forest w.r.t.*** D

If F is a deterministic forest with respect to D, then $\exists\, T_1, \dots, T_p$, p disjoint deterministic trees w.r.t. $D_{V_{T_1}}, \dots, D_{V_{T_p}}$ respectively, such that $\bigcup_{k=1}^{p} V_{T_k} = V$ and

$$F = \bigcup_{k=1}^{p} T_k.$$

Note: the '$\cup$' notation is extended to the canonical union for graphs, *i.e.*

$$G \cup G' = (V_G \cup V_{G'}, A_G \cup A_{G'}).$$

We now present the main theoretical contribution of this section, which extends Proposition 3 to deterministic forests. For a given forest F, we remind that the set of F's roots is denoted by

$$\mathcal{R}(F) = \{i \in V \mid \pi^F(i) = \emptyset\}.$$

Proposition 4 *Suppose F is a deterministic forest w.r.t. D. Let $G^*_{\mathcal{R}(F)}$ be a solution of the structure learning optimization problem introduced in Equation (1.9) with the MLL score w.r.t. $D_{\mathcal{R}(F)}$ (the subset of D containing observations of $\mathbf{X}_{\mathcal{R}(F)}$) i.e.*

$$s^{MLL}_{D_{\mathcal{R}(F)}}(G^*_{\mathcal{R}(F)}) = \max_{G \in DAG_{\mathcal{R}(F)}} s^{MLL}_{D_{\mathcal{R}(F)}}(G).$$

Then, $G^* = F \cup G^*_{\mathcal{R}(F)}$ *is a solution of the problem given in Equation* (1.9) *w.r.t.* D*, i.e.*

$$s_D^{MLL}(G^*) = \max_{G \in DAG_V} s_D^{MLL}(G).$$

Idea of proof The proof is detailed in Appendix A.2. Its main idea relies on the fact that, if F is a deterministic forest w.r.t. D, all the information associated with $\mathbf{X}$ is contained in $\mathbf{X}_{\mathcal{R}(F)}$, *i.e.*

$$H^D(\mathbf{X}) = H^D(\mathbf{X}_{\mathcal{R}(F)}).$$

Remarks Proposition 4 shows that the optimization problem for all nodes can be narrowed down to the optimization problem for roots of a deterministic forest only, if the criteria to be optimized is the MLL score.

This is a general result in the sense that, as opposed to Proposition 3, the assumptions of Proposition 4 are always verified. If there is no pairwise determinism in the dataset D, then $F_\emptyset = (V, \emptyset)$ is the only deterministic forest w.r.t. D, and solving problem given in Equation (1.9) for $G^*_{\mathcal{R}(F_\emptyset)}$ is the same as solving it for G^*.

Of course, Proposition 4 is all the more interesting that the number of roots of the deterministic forest F, $|\mathcal{R}(F)|$, is small compared to n. This enables us to focus on a smaller and easier structure learning problem while still having the guarantee to learn an optimal Bayesian network with regards to the MLL score.

Extreme example with a complete DAG $G^*_{\mathcal{R}(F)}$ For example, suppose that we regularize the structure learning problem by restricting the structure space DAG_V to $\{G \in DAG_V \mid \max_{1 \leq i \leq n} |\pi^G(i)| \leq P\}$ for $P << n$. If we are able to find F a deterministic forest w.r.t. D such that $|\mathcal{R}(F)| \leq P$, then the graph $G^* = F \cup G^{comp}_{\mathcal{R}(F)}$, with $G^{comp}_{\mathcal{R}(F)}$ any complete DAG on the roots, satisfies the constraints while maximizing the MLL score (from Proposition 4).

2.2.3 Extension to any deterministic DAG

The proof of Proposition 4 (Appendix A.2.) suggests that this property still holds when F is *any kind of deterministic DAG* (and not necessarily a forest). However,

1. Finding a deterministic DAG on a set of n random variables is NP-Hard since we need to test every combination of parents for every variable, which is not very interesting since our goal is to simplify the structure learning problem.

2. The density of a deterministic DAG is not known a priori, whereas we have absolute control over the number of parents per variables in the case of trees and forests.

3 Bayesian network structure learning with determinism screening

In this section, we detail the design of the `ds-BNSL` algorithm, that speeds up Bayesian network structure leaning in presence of pairwise deterministic relationships. For this purpose, we first define the notion of *redundancy* for categorical random variables and prove associated properties, that are necessary to guarantee that all of our proposed algorithms are well defined.
The proof of lemmas and propositions can be found in Appendix A.2.

3.1 Redundancy: definition, properties and preprocessing algorithm

3.1.1 Redundant variables

We define redundant variables, and introduce a set of associated properties.

Definition 6 ***Redundant variables***
For a given $n-$tuple of variables $\mathbf{X}$ and an associated dataset D, we define the relationship denoted by $\underset{D}{\leftrightarrow}$ as follows: for $i, j \in [\![1, n]\!]$,

$$X_i \underset{D}{\leftrightarrow} X_j \;\Leftrightarrow\; H^D(X_i|X_j) = 0 \text{ and } H^D(X_j|X_i) = 0.$$

*We will say that X_i and X_j are **redundant** with respect to D.*

The following Lemma states that if two variables are redundant, they have the same empirical entropy.

Lemma 7 *In the setting of Definition 6,*

$$X_i \underset{D}{\leftrightarrow} X_j \;\Rightarrow\; H^D(X_i) = H^D(X_j).$$

It is now straightforward to show that being redundant w.r.t. to a dataset D is an equivalence relationship, *i.e.* a *symmetric*, *reflexive* and *transitive* binary relationship.

Proposition 5 *The relationship $\underset{D}{\leftrightarrow}$ is an equivalence relationship on the set $\{X_1, \ldots, X_n\}$.*

We recall that an equivalence relationship defines a partition of the associated set, formed by the relationship **equivalence classes**.

3.1.2 Deterministic directed graphs, redundancy and cycles

We introduce the following intuitive lemma:

Lemma 8 *If X_i and X_j are categorical variables such that $H^D(X_i|X_j) = 0$, then*

$$|Val(X_i)| \leq |Val(X_j)| \tag{2.3}$$

with equality in Equation (2.3) if and only if $X_i \underset{D}{\leftrightarrow} X_j$.

The next result states that only redundant variables can cause the introduction of a cycle in a deterministic DG with at most one parent per node.

Proposition 6 *Let* $\mathbf{X}$ *a tuple of random variables indexed by* V*, and observed in a dataset* D*. Let* $G = (V, A)$ *a* ***directed graph*** *such that:*

$$\forall i \in V,\ |\pi^G(i)| \leq 1$$
$$\forall i \in V \ s.t.\ \pi^G(i) \neq \emptyset,\ H^D(X_i | X_{\pi^G(i)}) = 0.$$

Then, if there exists a cycle in G*, i.e.,*

$$\exists i_1, \dots i_p \in V,\ s.t.\ (i_1, i_2), (i_2, i_3), \dots, (i_{p-1}, i_p), (i_p, i_1) \in A,$$

the associated variables are all redundant (in the same equivalence class with respect to the relationship $\underset{D}{\leftrightarrow}$*).*

This proposition leads to the following result:

Proposition 7 *In the setting of Proposition 6, if there is no redundant variables in* $\mathbf{X}$*, then any directed graph* G *such that:*

$$\forall i \in V,\ |\pi^G(i)| \leq 1$$
$$\forall i \in V \ s.t.\ \pi^G(i) \neq \emptyset,\ H^D(X_i | X_{\pi^G(i)}) = 0$$

is a ***forest****.*

3.1.3 Handling redundancy: the `IdentifyRedundancy` algorithm

In practice, the risk of having redundant variables exists, especially in the context of IoT data. There are two main options to solve this issue, both of which imply a procedure in $\mathcal{O}(n^2)$ complexity:

1. **Discard every redundant variable** (keep only one representative by equivalence class for the equivalence relation $\underset{D}{\leftrightarrow}$). This should be considered if our priority is to have no redundant information in the final graph.

2. Establish a rule to **graphically represent each class of redundant variables**, in order to make sure no cycle is introduced. This approach enables all original variables of interest to be represented in the graph, even if some of them are redundant (*e.g.* two variables represent the same concept, but in different units, or different languages). It is the safest approach if we have no information concerning the variables that will be interesting later on.

We propose to follow the second approach with the `IdentifyRedundancy` algorithm, presented as Algorithm 3 below. The idea is to choose one representative by redundancy class, and make it the parent of all the other elements of its class in the final graph. This enables to keep all the original variables in the final graph, while making sure no cycle is introduced, as we only consider the chosen representatives as potential parents for the other variables.
Note that a very similar algorithm could be used for the first approach (discarding redundant variables).

This algorithm takes as only input the **empirical conditional entropy matrix** w.r.t. D, which we denote by $\mathbb{H}^D$, *i.e.*

$$\mathbb{H}^D = \left(H^D(X_i|X_j)\right)_{1\leq i,j\leq n}.$$

It returns an adjacency matrix $\mathbb{A}$ corresponding to the graphical representation of redundant data.

Algorithm 3 `IdentifyRedundancy`: Choose a representative for each group of redundant variables

Input: $\mathbb{H}^D$
$\mathbb{A} \leftarrow \left(\mathbb{I}_{\mathbb{H}^D_{ij}=0}\right)_{1\leq i,j\leq n}$
$I \leftarrow [\![1, n]\!]$ *#list of indexes to go through*
while $I \neq \emptyset$ **do**
  $i \leftarrow I[1]$
  $S_i \leftarrow \text{which}\left(\mathbb{A}_{i\bullet} \times \mathbb{A}_{\bullet i} == 1\right)$
  if $|S_i| > 1$ **then** *#there is at least one redundant variable with i*
    for $k \in S_i \setminus \{i\}$ **do**
      $\mathbb{A}_{ki} \leftarrow 0$ *#no other node from the class can be parent of i*
      for $l \in S_i \setminus \{k, i\}$ **do**
        $\mathbb{A}_{lk} \leftarrow 0$ *#nodes $\neq i$ from the class S_i cannot be connected*
  $I \leftarrow I \setminus S_i$
Output: $\mathbb{A}$

Going through `IdentifyRedundancy`

- The matrix $\mathbb{A}$, that is dynamically modified in the algorithm, represents a *potential adjacency matrix*. Each 1 in $\mathbb{A}$ represents an arc that can potentially be part of a deterministic forest. We start with $\mathbb{A} \leftarrow \left(\mathbb{I}_{\mathbb{H}^D_{ij}=0}\right)_{1\leq i,j\leq n}$, meaning that any empirically deterministic relationship can potentially be represented by an arc, then we select arcs in between variables that are equivalent with regards to the relation $\underset{D}{\leftrightarrow}$.
- We identify groups of variables that are redundant w.r.t. D. There are no principled way to choose the structure representation of these variables from data only. By convention, we choose the variable with the lowest index as the representative of the redundancy class: its

associated node is the parent of all the nodes associated with the other variables in the class. This is consistent with Proposition 3 since every child node is empirically deterministic given its parent node.

- Since $\underset{D}{\leftrightarrow}$ is an equivalence relationship, its equivalence classes form a partition of V. Let $i_1, \dots, i_p \in V$ such that $\{S_{i_1}, \dots S_{i_p}\}$ forms a partition of V. The complexity of this algorithm is $\mathcal{O}(\sum_{k=1}^{p} |S_{i_k}|^2)$, which is obviously $\mathcal{O}(n^2)$.
- Line 7, we make sure that no node k in S_i can be a potential parent of its chosen representative i. Every node in S_i is in the same equivalence class as k for the relation $\underset{D}{\leftrightarrow}$, but i being the smallest of the indexes, it is chosen as the parent of the other variables in our convention.

3.2 Choosing among deterministic trees

It is possible that several different deterministic trees coexist, as we will see in the next detailed example. In this case, we follow the rule of the fit-complexity tradeoff and choose to select the tree with the lowest number of parameters (which is a **complexity** criterion) among all deterministic trees (which all maximize the MLL score, which is a **fit** criterion).

3.2.1 Example of deterministic tree selection

Setting We consider $\mathbf{X} = (X_1, \dots, X_5)$, where the X_is are categorical, with respective value sets $Val(X_1), \dots, Val(X_5)$, with the following cardinalities: $|Val(X_1)| = 100$, $|Val(X_2)| = 20$, $|Val(X_3)| = 5$, $|Val(X_4)| = 10$, $|Val(X_5)| = 10$.
We suppose that we possess a dataset D containing M observations of $\mathbf{X}$, in which the following equations hold:

$$H^D(X_2|X_1) = 0,$$
$$H^D(X_4|X_2) = 0,$$
$$H^D(X_5|X_2) = 0,$$
$$H^D(X_3|X_1) = 0.$$

Note that this notably implies $H^D(X_4|X_1) = 0$ and $H^D(X_5|X_1) = 0$.
In this setting, there exists several deterministic trees with respect to D.

Deep deterministic tree Let $G_{deep} = (\{1,2,3,4,5\}, \{(1,2),(1,3),(2,4),(2,5)\})$, represented in Figure 2.2.[3]

Shallow deterministic tree Let $G_{shall} = (\{1,2,3,4,5\}, \{(1,2),(1,3),(1,4),(1,5)\})$, represented in Figure 2.3.

[3]For readability reasons, we label the nodes in the structure by the names of the variables they represent.

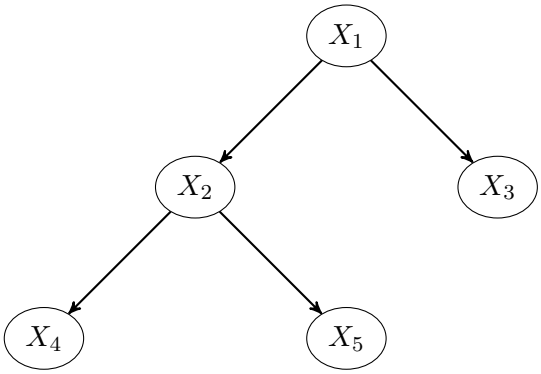


Figure 2.2: Representation of the tree G_{deep}

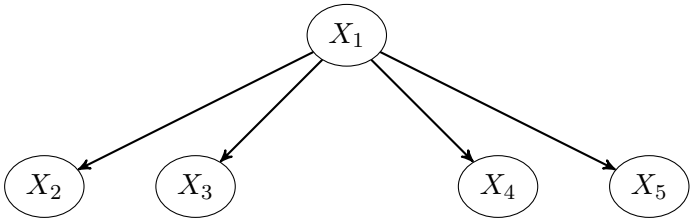


Figure 2.3: Representation of the tree G_{shall}

Selecting the less complex tree Considering the number of accessible values of variables $X_1, \ldots, X_5$, we have:

$$\begin{aligned}\mathcal{P}(G_{deep}) &= 100 \times 19 + 100 \times 4 + 20 \times 9 + 20 \times 9 \\ &= 2660, \\ \mathcal{P}(G_{shall}) &= 100 \times (19 + 4 + 9 + 9) \\ &= 4100.\end{aligned}$$

Therefore, G_{deep} is much simpler than G_{shall}, in terms of number of parameters.

Intuitive link between complexity, quantity of information, and depth Considering that the arcs of the trees represent pairwise deterministic relationships, the deep structure G_{deep} intuitively contains more information than the shallow structure G_{shall}. Indeed, the fact that $H^D(X_4|X_2) = 0$ is not visible in G_{shall}, whereas every deterministic relation encoded in G_{shall} is also visible in G_{deep}. Of course, in this particular case, we do not interpret DAG structures as representing conditional independence relations like we usually do with Bayesian networks. Indeed, we are facing (empirical) deterministic relationships, which make the faithfulness property fail, and interpretation in terms of conditional independence loose its value. The interpretation we are looking for in this deterministic setting is rather '*which variable is enough to know which other one*'. Deterministic trees such as G_{deep} and G_{shall} encode this information, in addition to also maximizing the MLL score.

3.2.2 Choosing the less complex deterministic tree: formalization and discussion

For a given tuple of variables $\mathbf{X}$ indexed by $V = [\![1, n]\!]$ and observed in a dataset D, we define $DT_V \subset DAG_V$ as the set of deterministic trees with respect to D.

We denote by $T_V = \{G \in DAG_V \mid \exists r \in V \ s.t. \ \forall i \in V \setminus \{r\}, |\pi^G(i)| = 1\}$ the set of trees with nodes V. We can write:

$$DT_V = \{G \in T_V \mid \forall (i,j) \in E_G, \ \mathbb{H}^D(X_i|X_j) = 0\}. \tag{2.4}$$

Moreover, thanks to Proposition 3 we have:

$$DT_V \subset \underset{G \in DAG_V}{\operatorname{argmax}} \ s_D^{MLL}(G).$$

In other words, the set of deterministic trees is a subset of the set of structures which maximize the MLL score (a subset containing only sparse structures).

Choosing the less complex deterministic tree can be formalized as the following optimization problem:

$$T^* \in \underset{G \in DT_V}{\operatorname{argmin}} \ \mathcal{P}(G), \tag{2.5}$$

where we remind that $\mathcal{P}(G)$ is the number of free parameters of a Bayesian network with structure G.

It should be noted that this is also equivalent to

$$T^* \in \underset{G \in DT_V}{\operatorname{argmax}} \ s_D^{BIC}(G).$$

As a general principle in ML, we want to optimize the complexity-fit tradeoff. Usually, this is done by optimizing an hyper parameter λ in a score defined by $(fit - \lambda \times complexity)$, such as the BIC score presented in Chapter 1. In the very particular case where there exists a deterministic tree however, we choose to look for the model that minimizes the total number of parameters (**complexity**) among a set of models that already maximize the MLL score (**fit**): the deterministic trees.

Link between complexity and depth for deterministic trees Choosing the deterministic tree with the smallest number of parameters is all the more motivated in this case, by the fact that the number of parameters of a deterministic tree is linked to its depth, and therefore to the model's **interpretability**, as we have seen in the previous example.
This correlation can be proven in the following illustrative setting: assume a collection of random variables $X_1, \dots X_k, X_{k+1}$, such that $H^D(X_i|X_{i-1}) = 0$ for all $i \in \{2, \dots, k+1\}$. This

notably implies that X_{k+1} has all the variables $X_1, \dots X_k$ as potential deterministic parents[4] Thanks to Lemma 8, we know that these hypotheses imply that $(|Val(X_i)|)_{1 \leq i \leq k+1}$ is decreasing. Therefore, minimizing the number of parameters implies to choose X_k as X_{k+1}'s parent variable. By immediate induction, we see that the less complex deterministic tree modeling variables $\{X_1, \dots, X_{k+1}\}$ is:

$$T = (\{1, \dots, k+1\}, \{(i, i+1)\}_{1 \leq i \leq k}),$$

which is indeed the deepest deterministic tree with nodes $[\![1, \dots, k+1]\!]$.

3.2.3 Choosing parents independently: a sound approach if no redundant variables are present

Decomposability of the parameter-minimizing problem We recall that, for $i \in V$, G_i denotes the local subgraph of G centered on i, defined in Equation (1.6) in Chapter 1. For any DAG G, the total number of parameters $\mathcal{P}(G)$ decomposes as the sum of the local number of parameters $\mathcal{P}(G_i)$, *i.e.*

$$\mathcal{P}(G) = \sum_{i=1}^{n} \mathcal{P}(G_i). \tag{2.6}$$

Independently choosing the simplest deterministic parent variable for each variable is not guaranteed to return a DAG: there is a risk of introducing cycles. However, assuming *there are no redundant variables in* D, Proposition 7 tells us that a deterministic directed Graph w.r.t. D is necessarily acyclic. In that case, one may choose $\pi^G(i)$ independently for each $i \in V$, as long as $H^D(X_i | X_{\pi^G(i)}) = 0$, without risking to introduce a cycle in G. Therefore, minimizing $\mathcal{P}(G)$ among a set of deterministic DAGs *narrows down to minimizing independently each term* $\mathcal{P}(G_i)$ appearing in the right hand side of Equation (2.6).

3.2.4 Finding the best local deterministic tree in practice: the `BestParent` algorithm

We recall that the number of parameters of a local distribution $X_i \mid \mathbf{X}_{\pi^G(i)}$ is given by:

$$\mathcal{P}(G_i) = (|Val(X_i) - 1|) \times |Val(\mathbf{X}_{\pi^G(i)})|.$$

Therefore, if a given variable X_i has many potential deterministic parent variables, minimizing the number of parameters of the local distribution $X_i | X_{\pi^G(i)}$ narrows down to choosing the parent $\pi^G(i)$ of i such that $X_{\pi^G(i)}$ has the smallest number of values $|Val(X_{\pi^G(i)})|$ among all potential deterministic parent variables.

This leads to the Algorithm 4, which takes as input:

[4] This is a common setting in practice when considering variables that all correspond to the same descriptive dimension (location, data type, etc), as there tends to be a inherent hierarchy among those variables in terms of information granularity.

- $i \in V$: node for which we want to select the best parent,
- $\pi_{pot}(i) \subset V \setminus \{i\}$: set of potential parents for i we want to select from,
- D: dataset with columns indexed by V,

and returns $\pi(i)$, a proposed parent for variable i.

Algorithm 4 `BestParent`: Best single parent selection

Input: i, $\pi_{pot}(i)$, D

1: $\pi(i) \leftarrow \underset{j \in \pi_{pot}(i)}{\text{argmin}} |Val(X_j)|$ [1]

Output: $\pi(i)$

Remarks on `BestParent`

- The '[1]' means that if $\underset{j \in \pi_{pot}(i)}{\text{argmin}} |Val(X_j)|$ contains more than one node, we choose the one with the lowest index by convention. In that case, there is no way to statistically discriminate variables belonging to $\{X_k,\ k \in \underset{j \in \pi_{pot}(i)}{\text{argmin}} |Val(X_j)|\}$ with regards to their relation with the considered variable X_i. They all guarantee an optimal local MLL score, with a minimal number of parameters.
- `BestParent` is at worst $\mathcal{O}(n)$, since we only need to go through the set $\pi_{pot}(i)$ once to find the less complex parent[5], and we obviously have $|\pi_{pot}(i)| \leq n$.

3.3 Determinism screening: finding the optimal deterministic forest

3.3.1 Minimizing $\mathcal{P}(F)$ over all deterministic forests F

As explained in Section 3.2.3, under the non-redundancy assumption, searching for the less complex deterministic DAG among a set of deterministic DAG comes down to solving independent local parent search problems for each node $i \in V$. In the case of forests, we are only searching for single deterministic parents: for each i, we choose the deterministic parent $\pi^G(i)$ which minimizes $|Val(X_{\pi^G(i)})|$.

3.3.2 Presenting the `DeterScreen` algorithm

Since we are able to identify and handle redundancies efficiently using `IdentifyRedundancy`, we can now propose the following `DeterScreen` algorithm, that finds the less complex deterministic forest w.r.t. to a dataset D. Proposition 7 guarantees that the returned graph is indeed a forest.

[5] We consider that the information regarding the number of accessible values for each random variable X_i, $i \in V$ is accessible in constant time from D, as this information is stored jointly with the data in most high-level programming languages (as it is the case with `dataframes` in `R`).

This algorithm takes as only input D, a dataset containing M observations of $\mathbf{X}$, and returns a deterministic forest F.

Algorithm 5 `DeterScreen`: Determinism screening

Input: D

$\mathbb{H}^D \leftarrow \left(H^D(X_i|X_j)\right)_{1\leq i,j\leq n}$

$\mathbb{A}^D \leftarrow$ `IdentifyRedundancy`$(\mathbb{H}^D)$

for $i = 1$ to n **do** *#choose the best parent from the set of potential parents*

  $\pi_{pot}(i) \leftarrow \{j \in V \setminus \{i\} \mid \mathbb{A}^D_{ij} = 0\}$

  $\pi^*(i) \leftarrow$ `BestParent`$(i, \pi_{pot}(i), D)$

$F \leftarrow (V, \{(\pi^*(i), i) \mid i \in V \ s.t. \ \pi^*(i) \neq \emptyset\})$

Output: F

Complexity overview `DeterScreen` has a quadratic worst-case complexity:

- computing $\mathbb{H}^D$ is doable in $\mathcal{O}(n^2)$ operations,
- `BestParent` ($\mathcal{O}(n)$) is called at most n times,
- `IdentifyRedundancy` has $\mathcal{O}(n^2)$ complexity.

3.4 Bayesian network structure learning with determinism screening: the `ds-BNSL` algorithm

We now propose the Bayesian network structure learning with determinism screening (`ds-BNSL`) algorithm (Algorithm 6). This algorithm exploits determinism screening and the intuition given by Proposition 4 to narrow the structure learning methods down to a subset of the original variables (the roots of the deterministic forest found by the `DeterScreen` algorithm).

This algorithm takes as input:

- D: a dataset containing M observations of $\mathbf{X}$,
- `sota-BNSL`: a standard Bayesian network structure learning algorithm (typically close to state of the art), taking a dataset as input, and returning a Bayesian network structure.

Algorithm 6 `ds-BNSL`: Bayesian network structure learning with determinism screening

Input: D, `sota-BNSL`

$F \leftarrow$ `DeterScreen`(D)

Identify $\mathcal{R}(F) = \{i \in V \mid \pi^F(i) = \emptyset\}$, the set of F's roots.

$G^*_{\mathcal{R}(F)} \leftarrow$ `sota-BNSL`$(D_{\mathcal{R}(F)})$

$G^* \leftarrow F \cup G^*_{\mathcal{R}(F)}$

Output: G^*

Complexity: `ds-BNSL` is theoretically faster than `sota-BNSL` in presence of determinism This algorithm can be interpreted as a pre-processing of a standard Bayesian network structure learning algorithm, which enables faster learning of the final structure in the presence of deterministic relationships. Indeed,

- Line 1 has $\mathcal{O}(n^2)$ complexity,
- Line 2 has $\mathcal{O}(n)$ complexity,
- Line 4 has $\mathcal{O}(n)$ complexity.

Line 3 potentially has a far greater complexity, since it corresponds to the application of a standard Bayesian network structure learning algorithm, which are known to be time intensive. However, this algorithm is only applied to a subset of the original dataset. We therefore have every reason to believe that this algorithm could significantly accelerate the structure learning task as long as $|\mathcal{R}(F)| < n$.
We will conduct a more detailed complexity analysis in the next section.

Performance: MLL score is not the final goal, but `ds-BNSL` is still promising To have the guarantee that this algorithm learns the optimal structure with regards to the MLL score, we would need the algorithm `sota-BNSL` to learn an optimal structure regarding this score (Proposition 4). However, we have seen that maximizing the MLL score is not a good goal in general since it overfits. We only accept it as an objective when we are learning trees, which are sparse structures. The structure learning algorithm `sota-BNSL` should in practice be chosen among standard structure learning algorithms that have been proven to model data accurately. The guarantees concerning the final score of the learned structure (as those presented in Propositions 3 and 4) do not hold anymore, but the underlying idea stays the same: in the presence of deterministic relationships, even though the MLL score is not our goal, we propose to first identify the best deterministic forest F, then to narrow down the structure learning task to a subset of the original variables: the roots of F.

We will show in the next section how this idea can be extended to any kind of data (that do not necessarily contain deterministic relationships).

4 Extension to generic data: strong pairwise relationships screening

In this section, we propose an extension of the notion of determinism to the one of **quasi-determinism**, from which we derive the `QuasiDeterScreen` algorithm, that screens not only pairwise deterministic relationships but more generally pairs of strongly related variables.

We then present `qds-BNSL`, which generalizes `ds-BNSL`, study its complexity, and show that it is very promising in terms of computational performance compared to standard Bayesian Network structure learning algorithms on generic (a priori) non-deterministic data.

In **all this section**, we consider that $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a tuple of categorical random variables indexed by $V = [\![1, n]\!]$, and that D is a complete dataset containing M observations of $\mathbf{X}$.
The proof of all lemmas and propositions can be found in Appendix A.2.

4.1 Quasi-determinism

In many datasets, one does not observe true empirical determinism, although there are still very strong relationships between some of the variables. We therefore propose to *relax* the notion of determinism to quasi-determinism, where *quasi* is meant with respect to an hyperparameter ε: we talk about $\varepsilon-$*quasi-determinism*.

There are several ways to measure how close a relationship is from deterministic. Huhtala et al. (1999) consider the minimum number of observations that must be dropped from the data for the relationship to be empirically deterministic. Since we are in a score-maximization context, we will rather use ε as a threshold on the empirical conditional entropy. The following definition is the natural extension of Definition 3.

Definition 7 *$\varepsilon-$**quasi-determinism** ($\varepsilon-$**qd**)*
*Given a dataset D containing observations of variables X_i and X_j, the relationship $X_i \to X_j$ is $\varepsilon-$quasi-deterministic ($\varepsilon-$**qd**) w.r.t. D iff $H^D(X_j|X_i) \leq \varepsilon$.*

It has been seen in Proposition 4 that a deterministic forest is a (sparse) subgraph of an optimal DAG with respect to the MLL score. 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 $\varepsilon-$qd criteria presented in Definition 7, we now propose the quasi-determinism screening approach to Bayesian network structure learning.

4.2 Quasi-determinism screening algorithm

Algorithm 7 (`QuasiDeterScreen`) details how to find the simplest $\varepsilon-$qd forest F_ε from a dataset D and a threshold ε. As the case of deterministic forests, *simplest* refers to the complexity in terms of number of parameters of F_ε: $\mathcal{P}(F_\varepsilon)$.

This algorithm takes as input:

- D: a dataset containing M observations of $\mathbf{X}$,
- ε: a threshold for quasi-determinism.

Algorithm 7 `QuasiDeterScreen`: Quasi-determinism screening

Input: D , ε

1: $\mathbb{H}^D \leftarrow \left(H^D(X_i|X_j)\right)_{1\leq i,j\leq n}$
2: $\mathbb{A}^{D,\varepsilon} \leftarrow \left(\mathbb{I}_{\{\mathbb{H}^D\leq\varepsilon\}}\right)$
3: **for** $i = 1$ to n **do** *#check for cycles in $\varepsilon-$qd relations*
4: $\quad S_i \leftarrow \texttt{which}\left(\mathbb{A}^{D,\varepsilon}_{i\bullet} \times \mathbb{A}^{D,\varepsilon}_{\bullet i} == 1\right)$
5: $\quad$ **if** $|S_i| > 1$ **then**
6: $\quad\quad$ **for** $j \in S_i \setminus \{i\}$ **do**
7: $\quad\quad\quad$ **if** $\mathbb{H}^D_{ji} \leq \mathbb{H}^D_{ij}$ **then** *#remove the arc corresponding to the smallest conditional entropy*
8: $\quad\quad\quad\quad \mathbb{A}^{D,\varepsilon}_{ij} \leftarrow 0$
9: $\quad\quad\quad$ **else**
10: $\quad\quad\quad\quad \mathbb{A}^{D,\varepsilon}_{ji} \leftarrow 0$
11: **for** $i = 1$ to n **do** *#choose the simplest among all potential parents*
12: $\quad \pi^{\varepsilon}_{pot}(i) \leftarrow \texttt{which}\left(\mathbb{A}^{D,\varepsilon}_{i\bullet} == 1\right)$
13: $\quad \pi^{\varepsilon*}_{pot}(i) \leftarrow \texttt{BestParent}(i, \pi^{\varepsilon}_{pot}(i), D)$
14: $F_\varepsilon \leftarrow (V_{F_\varepsilon}, A_{F_\varepsilon})$ where $V_{F_\varepsilon} = [\![1, n]\!]$ and $A_{F_\varepsilon} = \{(\pi^{\varepsilon*}_{pot}(i), i) \mid i \in [\![1, n]\!] \; s.t. \; \pi^{\varepsilon*}_{pot}(i) \neq \emptyset\}$

Output: F_ε

Specificities of `QuasiDeterScreen` compared to `DeterScreen` We cannot proceed the same way as in `IdentifyRedundancy` because of the fact that the relation $\underset{\varepsilon,D}{\leftrightarrow}$, defined as

$$X_i \underset{\varepsilon,D}{\leftrightarrow} X_j \;\Leftrightarrow\; \max\left(H^D(X_i|X_j), H^D(X_j|X_i)\right) \leq \varepsilon$$

is *not* transitive *a priori*, and hence is not an equivalence relationship.

We cannot form a partition of V with redundancy classes like we did in the previous section, and are therefore forced to have a routine a bit more complex (from lines 3 to 10) that goes through every node in V and considers S_i for each one of them.

This routine uses a test (on line 7): $\mathbb{H}^D_{ij} \leq \mathbb{H}^D_{ji}$ which evaluates if X_i is better explained by X_j than X_j is explained by X_i. It then keeps only the arc corresponding to the relationship judged the most significant.

The overall complexity of this routine is still $\mathcal{O}(n^2)$, even though we actually are forced to go through more elements that it was the case when using `IdentifyRedundancy`. However, we are loosing some interesting theoretical guarantees, as the fact that the $\varepsilon-$quasi-deterministic forest that is returned by `QuasiDeterScreen` is not necessarily the one which maximizes the BIC score among all the $\varepsilon-$quasi-deterministic forests.

4.3 Learning Bayesian networks using quasi-determinism screening

We now present Algorithm 8 (`qds-BNSL`), which uses quasi-determinism screening to accelerate Bayesian network structure learning. This algorithm takes as input:

- D: a dataset containing M observations of $\mathbf{X}$,
- ε: a threshold for quasi-determinism,
- `sota-BNSL`: a standard Bayesian network structure learning algorithm (typically close to state of the art), taking a dataset as input, and returning a Bayesian network structure.

Algorithm 8 `qds-BNSL`: Bayesian network structure learning with quasi deterministic screening

Input: D, ε, `sota-BNSL`
1: $F_\varepsilon \leftarrow$ `QuasiDeterScreen`(D, ε)
2: Identify $\mathcal{R}(F_\varepsilon) = \{i \in [\![1, n]\!] \mid \pi^{F_\varepsilon}(i) = \emptyset\}$, the set of F_ε's roots.
3: $G^*_{\mathcal{R}(F_\varepsilon)} \leftarrow$ `sota-BNSL`$(D_{\mathcal{R}(F_\varepsilon)})$
4: $G^*_\varepsilon \leftarrow F_\varepsilon \cup G^*_{\mathcal{R}(F_\varepsilon)}$
Output: G^*_ε

The extension of the definition of determinism to quasi-determinism (Definition 7) prevents us to have guarantees as those presented in Proposition 4. However, under some conditions on `sota-BNSL`, we are able to get explicit bounds for the MLL score of a graph G^*_ε returned by `qds-BNSL`, as stated in the following proposition.

Proposition 8 *Let ε, D and `sota-BNSL` be valid inputs to Algorithm 8, and G^*_ε 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 $s_D^{MLL}(G^*_\varepsilon)$:

$$s_D^{MLL}(G^*_\varepsilon) \geq \left(\max_{G \in DAG_V} s_D^{MLL}(G) \right) - Mn\varepsilon.$$

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

$$s_D^{MLL}(G^*_\varepsilon) \underset{\varepsilon \to 0}{\longrightarrow} \max_{G \in DAG_V} s_D^{MLL}(G).$$

In other words, $\varepsilon \mapsto s_D^{MLL}(G^*_\varepsilon)$ is continuous in 0, and Proposition 8 generalizes Proposition 4.

Let us now proceed to an analysis of the complexity of the quasi-determinism screening approach.

4.4 Complexity analysis

Complexity of `sota-BNSL` 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 $C_{sota}(n)$ be its complexity.
$C_{sota}(n)$ should typically be thought of as exponential, or at least high degree polynomial, in n.

Complexity of `QuasiDeterScreen` We have the following decomposition of the complexity of Algorithm 8:

1. Lines 1-2: $\mathcal{O}(n^2)$. Computation of $\mathbb{H}^D$: we need counts for every couple (X_i, X_j) for $i < j$ (each time going through all rows of D), which results in $M\frac{n(n-1)}{2}$ operations.
2. lines 3-10: $\mathcal{O}(n^2)$. Going through all elements of $\mathbb{H}^D$ once.
3. lines 11-13: $\mathcal{O}(n^2)$. Going through all elements of $\mathbb{H}^D$ once.

Therefore, overall $C_{qds}(n) = \mathcal{O}(n^2)$.

Complexity of `qds-BNSL` For a given dataset D,

$$\forall \varepsilon \geq 0,\ n_r(\varepsilon) = |\mathcal{R}(\texttt{QuasiDeterScreen}(D, \varepsilon))|.$$

The function $n_r(\cdot)$, associates to $\varepsilon \geq 0$ the number of roots of the forest F_ε returned by `QuasiDeterScreen` ran with inputs D and ε. The complexity of `qds-BNSL` then decomposes as:

1. Line 1: $\mathcal{O}(n^2)$. Execution of `QuasiDeterScreen`.
2. Lines 2-4: $C_{sota}(n_r(\varepsilon))$. Execution of `sota-BNSL` on the reduced dataset $D_{\mathcal{R}(F_\varepsilon)}$ containing observations of $n_r(\varepsilon)$ variables.

This yields $C_{qdsBNSL}(n) = \mathcal{O}(n^2) + C_{sota}(n_r(\varepsilon))$.

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

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

It is hard to obtain a closed-form of the difference $C_{sota} - C_{qdsBNSL}$, since it is not clear how to estimate the complexity of state of the art learning algorithms. However, we know that all Bayesian network structure learning algorithms are very time-intensive: $C_{sota}(n)$ is known to be typically exponential in n for the best exact structure learning algorithms, as those presented by Silander and Myllymäki (2006) or Cussens (2011), and it is expected to be significantly larger than $\mathcal{O}(n^2)$ for high-performing heuristics. We therefore expect an important decrease in computational time for `qds-BNSL` compared to `sota-BNSL`, as long as $n_r(\varepsilon)$ is sufficiently smaller than n. If moreover this is true for small[6] ε, we anticipate a small performance loss.

[6] We will explain in Chapter 4 what we mean by 'small' in this case.

5 Experiments

In this section, we present some experiments conducted both using data extracted from an IoT system backed by a relational database, and benchmark datasets used in the Bayesian network structure learning literature, which contain no empirical determinism.

5.1 Setting

5.1.1 Programming details

Bayesian network manipulation, and standard routines for learning and inference were programmed into wrappers functions relying on the `R` package `bnlearn` from Scutari (2010), which is a very good reference among open-source packages dealing with Bayesian networks structure learning.

The remaining of the code, including all of the algorithms presented in this chapter, was mostly written in `R`.
Challenges that were faced include:

- writing the `CVLL` score evaluation function without storing large objects into memory,
- optimization of entropy computation using large sparse matrices.

5.1.2 Choice of `sota-BNSL`

We need to pick a Bayesian network structure learning algorithm both to obtain a baseline performance, and to use after the screening phase of algorithms `ds-BNSL` and `qds-BNSL`. 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 (2005). In this section, we refer to this algorithm as `sota-BNSL`.

5.1.3 Algorithms evaluation

The algorithms are evaluated using 3 axes:

- **Performance**:
  - BDeu score presented in Chapter 1 with equivalent sample size (ESS) equal to 5, inspired from Teyssier and Koller (2005)
  - CVLL score presented in Chapter 1 with 10 folds.

- **Readability**: Number of arcs of the learned Bayesian network.
  The BDeu score naturally penalizes overly complex models (in terms of number of parameters), it is 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, which is therefore linked to how interpretable this structure is.

- **Computing time**: t_{run} (all algorithms were run on independent threads of the same machine, with an Intel CPU E5-2650 v2 2.60GHz with 8 nodes and 64Go of RAM).
  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.

5.2 Running the `ds-BNSL` algorithm on an IoT dataset

In Section 2, we proved results bridging the gap between score and search Bayesian network structure learning and the notion of empirical determinism. Then, in Section 3, we designed an algorithm exploiting determinism in data to accelerate the Bayesian network structure learning task. As it has been said, this algorithm is all the more interesting in practice as there actually are deterministic relationships between variables observed in the data. In this subsection, we present experiments we conducted on real datasets that natively contain such deterministic relationships, extracted from Schneider Electric's IoT systems.

5.2.1 Data

The experiments of this subsection have been conducted on descriptive metadata from the `HOMES` programme[7]. The full `HOMES` metadataset contains **1000** rows and **47** columns, several of which are keys resulting from the storage in a relational database. Notably, the `DATA_KEY` variable has a different value for every row in the dataset, and therefore has no real statistical significance.

In our experiments, we choose to use three datasets extracted from the `HOMES` metdataset:

1. The full dataset D_1, containing all **47** variables, including `DATA_KEY`. This case does not make a lot of statistical sense, but represents well what we would be facing if we wanted to learn a Bayesian network structure on unknown metadata without doing any preprocessing.

2. The sub-dataset D_2 containing **43** variables: all but the 4 variables with more than 750 values (table keys).

[7] `http://www2.schneider-electric.com/documents/press-releases/fr/shared/2011/11/20111123_dp_HOMES_FR.pdf`

3. The sub-dataset D_3 containing **41** variables: all but the 6 variables with more than 500 values.

It has to be noted that in D_1, there actually exists a deterministic tree, as in the hypothesis of Proposition 3 (we therefore know that this tree maximizes the MLL score). In D_2 and D_3, more and more deterministic relationships are dropped as we remove the variables that have the most accessible values. However, there still remains many deterministic relationships inherent to the data and the way it is stored.
Table 2.1 summarizes the sizes and number of deterministic trees in the best deterministic forest[8] for each of those three datasets.

Table 2.1: Presentation of the three datasets extracted from `HOMES` metadata. For $i \in \{1, 2, 3\}$, M and n are respectively the number of rows and columns of D_i, and $n_r(\varepsilon = 0)$ represents the number of roots of a deterministic forest w.r.t. D_i.

name	n	M	$n_r(\varepsilon = 0)$
D_1	47	1000	1
D_2	43	1000	5
D_3	41	1000	6

5.2.2 Results presentation and remarks

Tables 2.2 display the different algorithm evaluation criteria for `sota-BNSL` and `ds-BNSL`: computation time, BDe score, CVLL score and number of arcs. The CVLL score is not computable for the full metadataset D_1: the variable `DATA_KEY` systematically has all its values in the validation sets that are not observed in the corresponding training sets, and the convention for the computation of the CVLL score in that case is to discard all validation sets observations containing unobserved values of at least one of the variables (see Chapter 4 for further details). Note that in the case of `sota-BNSL` for D_1, the algorithm was unable to reach a local maximum of the BDe score before running out of memory and being forced to stop while computing the local score (as explained in Section 1.2). This explains why the graph learned on dataset D_1 is significantly sparser than the graphs learned with `sota-BNSL` on D_2 and D_3.

Figure 2.4 displays a visual comparison of the structures learned with `ds-BNSL` algorithm and the baseline `sota-BNSL` on D_1, D_2 and D_3. The legend associated with the variables is available in in Appendix B.

[8] Found with exhaustive search using the `DeterScreen` algorithm

dataset	`sota-BNSL`	`ds-BNSL`
D_1	$\mathbf{-68.76}$	-92.74
D_2	$\mathbf{-25.56}$	-42.18
D_3	$\mathbf{-18.20}$	-25.15

(a) BDeu score per sample of the learned structures

dataset	`sota-BNSL`	`ds-BNSL`
D_1	NaN	NaN
D_2	-8.72	$\mathbf{-8.62}$
D_3	-8.03	$\mathbf{-7.56}$

(b) CVLL score per sample of the learned structures

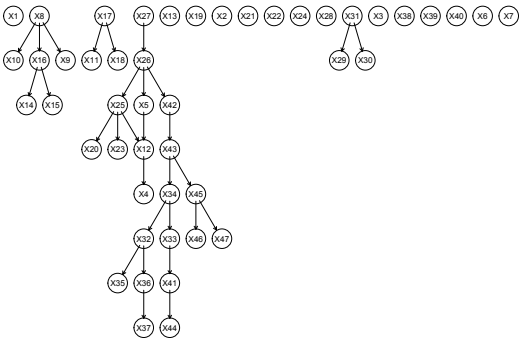
dataset	`sota-BNSL`	`ds-BNSL`
D_1	$92,406$	$\mathbf{3}$
D_2	$13,794$	$\mathbf{121}$
D_3	$4,346$	$\mathbf{3}$

(c) Algorithm computation time (seconds) of the algorithms

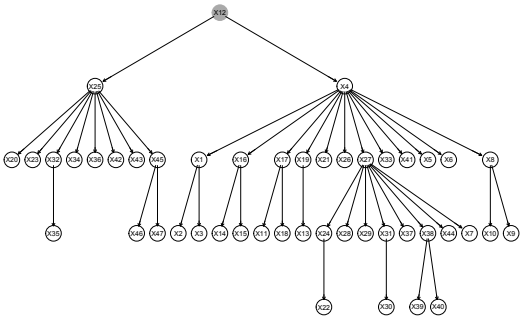
dataset	`sota-BNSL`	`ds-BNSL`
D_1	$\mathbf{30}$[9]	46
D_2	70	$\mathbf{41}$
D_3	102	$\mathbf{42}$

(d) Number of arcs of the learned structures

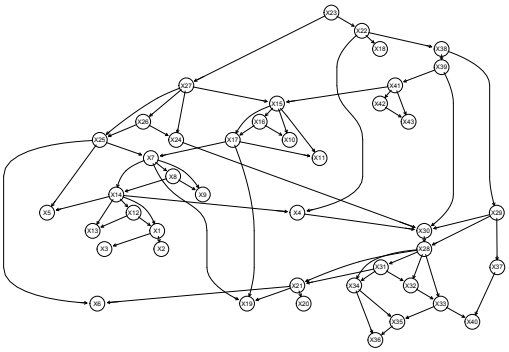
Table 2.2: Evaluation of the `ds-BNSL` algorithm and the baseline `sota-BNSL` algorithm on three datasets extracted from the `HOMES` metadataset, using several criteria: algorithm speed, graph's number of arcs, as well as BDeu and CVLL scores



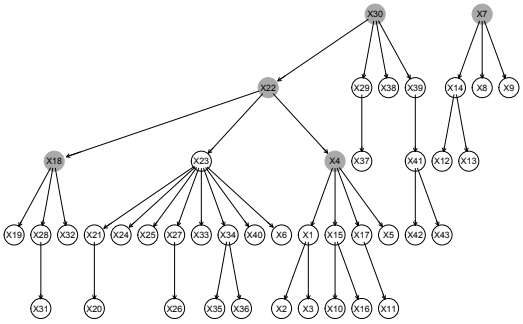
(a) BN learned with `sota-BNSL`[10] on D_1



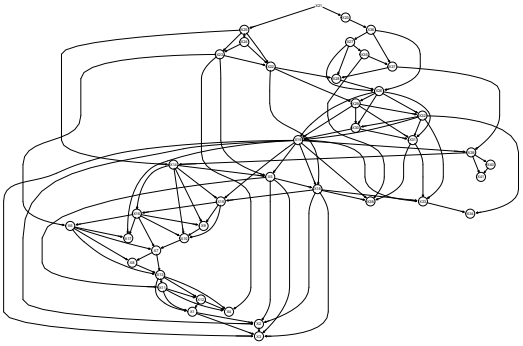
(b) BN learned with `ds-BNSL` on D_1



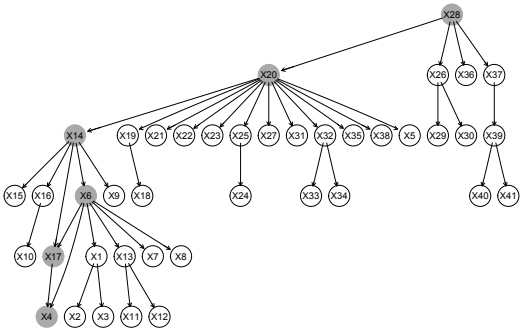
(c) BN learned with `sota-BNSL` on D_2



(d) BN learned with `ds-BNSL` on D_2



(e) BN learned with `sota-BNSL` on D_3



(f) BN learned with `ds-BNSL` on D_3

Figure 2.4: Bayesian networks learned on the subsets D_1, D_2 and D_3 of the `HOMES` dataset

5.2.3 Results interpretation

Score First, we remark that in this extreme case where some of the variables have a very important number of accessible values with regards to the number of observations (even in D_3, some variables have more than $\frac{M}{5}$ accessible values), BDeu and CVLL are not always entirely consistent, despite the fact that the BDe score is generally used as a proxy for generalization performance. As we will see in the next section, these two scores have a much closer behaviour when tackling datasets with an important number of observations compared to the variables' number of values.
Here, the BDeu score is consistently better for graph structures reached with our selected baseline algorithm `sota-BNSL` compared to graphs learned with `ds-BNSL`. This can be explained by the fact that arcs that are pre-screened by the `DeterScreen` procedure, and that appear in graphs returned by `ds-BNSL`, tend to be highly penalized by the BDe score because of the number of parameters they add to the model: this is a result of the number of values of variables that are typically deterministic parents.
This shows the limits of the BDe score: it seems to be a bad proxy for generalization performance in presence of deterministic relations implying variables with a high number of values.

Number of arcs The graphs learned with `ds-BNSL` are generally sparser that those learned with `sota-BNSL`.[11] This is not surprising, as this was one of the motivations behind the deterministic screening idea: the graphs learned with `ds-BNSL` are the union of a deterministic forest, which is very sparse (less than an arc per node) and a graph learned with `sota-BNSL` and the BDe score, which can be expected to have on average the same density of arcs per nodes than the graphs leaned with `sota-BNSL` on the entire dataset (which is closer to two arcs per node).

Computation time Computation time is what makes the potential of the screening algorithm when facing IoT data the most obvious, and was the main motivation behind the idea of the `DeterScreen` algorithm. The $\mathcal{O}(n^2)$ screening phase is, as expected, very fast compared to the baseline `sota-BNSL` algorithm, especially since variables that have a lot of values make the scores harder to compute.
In practice, when facing a new dataset containing unknown variables, `ds-BNSL` is therefore extremely interesting to try first, considering how little time it costs compared to directly running a standard heuristic. Moreover, the `DeterScreen` routine of `ds-BNSL`, which only has a quadratic time complexity, is a very efficient way to discover a possible deterministic structure between the variables (for example resulting from a relational or semantic database extraction), which could bring a lot of interesting insights in terms of preprocessing. For example: which variables in fact correspond to keys of sub-tables ? Do we want to keep these variables in the final structure ?

[11] This is not the case for D_1, but is obviously due to the premature end of `sota-BNSL`.

Which variables are redundant ? Is there a variable that has so many accessible values that it actually is an empirical deterministic parent to all the other variables ?

Visual perception The structure of the graph learned with `ds-BNSL` is obviously a lot more readable, as it naturally makes the structure underlying the variables appear. Even on dataset D_3, where most of the main key variables are missing, there still are natural 'clusters' that are formed.
An expert that is very familiar with the dataset and the relations between the variables is expected to typically give as input (*i.e.* knowledge elicitation) a graph were the subtables (and more generally every 'most simple' pairwise deterministic relationship) appears. In a sense, we could consider that the `DeterScreen` algorithm is a way to automatize the expert knowledge elicitation phase, by automatically detecting pairwise empirical deterministic relationships. In that sense, even if Propositions 3 and Proposition 4 presented previously in this chapter give theoretical insight to the `ds-BNSL` algorithm, one might consider that the goal behind determinism screening is not only score maximization, but also automatic knowledge elicitation. The deterministic forest (that contains only deterministic relationship) is then used as a starting point for a standard structure learning heuristic `sota-BNSL`, with the constraint that only the roots are considered for this second phase (theoretically sound with the MLL score and enabling a huge computational time gain).

5.3 Running the `qds-BNSL` on benchmark datasets

In this subsection we picked the largest of the benchmark datasets traditionally used by the Bayesian network structure learning community, none of which contain exact empirical determinism. We also considered a dataset from an online prediction challenge from the `DrivenData` website[12], which contains some variables with a high number of values and a few empirical deterministic relationships, although nothing comparable to the datasets used in the last subsection that contain determinism 'by design'.

5.3.1 Data

Table 2.3 summarizes the data used in our experiments. We considered the largest open-source categorical datasets among those presented[13] by Davis and Domingos (2010) and available on the UCI repository (Dheeru and Karra Taniskidou, 2017): 20 Newsgroup, Adult, Book, Covertype, KDDCup 2000, MSNBC, 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[14],

[12] `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/`
[13] `http://alchemy.cs.washington.edu/papers/davis10a/`
[14] `http://www.bnlearn.com/bnrepository/`

for each of which we simulated 10000 observations: Andes, Hailfinder, Hepar 2, Link, Munin 1-4, PathFinder and Win95pts.

Table 2.3: Datasets presentation

name	**short name**	n	M
20 newsgroups	20ng	930	11293
adult	adult	125	36631
book	book	500	8700
covertype	covertype	84	30000
kddcup 2000	kddcup	64	180092
msnbc	msnbc	17	291326
msweb	msweb	294	29441
plants	plants	69	17412
pump it up challenge	pumpitup	28	54000
reuters 52	r52	941	6532
uscensus	uscensus	68	2458285
andes	andes	223	10000
hailfinder	hailfinder	56	10000
hepar 2	hepar2	70	10000
link	link	724	10000
munin 1	munin1	186	10000
munin 2	munin2	1003	10000
munin 3	munin3	1041	10000
munin 4	munin4	1038	10000
pathfinder	pathfinder	109	10000
windows 95 pts	win95pts	76	10000

5.3.2 Choice of ε for `qds-BNSL`

An approach to choosing ε in the case of the *qds-BNSL* algorithm is to pick values for $n_r(\varepsilon)$, and manually find the corresponding values for ε. For a given dataset and $x \in [0, 1]$, we define $\varepsilon_x = n_r^{-1}(\lfloor xn \rfloor)$. In other words, ε_x is the value of ε for which the number of roots of the qd forest F_ε represents a proportion x of the total number of variables.
The computation of ε_x is not problematic: once $\mathbb{H}^D$ is computed and stored, evaluating $n_r(\varepsilon)$ is done in constant time, and finding one of $n_r(\cdot)$'s quantiles is doable in $\mathcal{O}(\log(n))$ operations (dichotomy), which is negligible compared to the overall complexity of the screening.

5.3.3 Results (1/2). Independent consideration of performance, readability and computation time

We present the obtained results for our selected baseline algorithm `sota-BNSL`, and 3 versions of `qds-BNSL`. For each dataset, we selected $\varepsilon \in \{\varepsilon_{0.9}, \varepsilon_{0.75}, \varepsilon_{0.5}\}$), corresponding to a restriction of `sota-BNSL` to 90%, 75% and 50% of the original variables respectively.

The results are shown in Tables 2.4, 2.5, 2.6 and 2.7, 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 (**qds**$_{\varepsilon_{0.9}}$, **qds**$_{\varepsilon_{0.75}}$ and **qds**$_{\varepsilon_{0.5}}$).

Table 2.4: BDeu score per sample. Every result that is less than 5% smaller than `sota-BNSL`'s score is boldfaced.

dataset	**sota**	**qds**$_{\varepsilon_{0.9}}$ (%)	**qds**$_{\varepsilon_{0.75}}$ (%)	**qds**$_{\varepsilon_{0.5}}$ (%)
20ng	-142.71	**-0.66**	**-2.13**	**-4.78**
adult	-12.86	**-0.16**	**-0.05**	**-4.01**
book	-34.81	**-0.80**	**-1.69**	**-4.64**
covertype	-13.60	**-0.21**	**-1.23**	-11.7
kddcup	-2.38	**-0.31**	**-1.04**	**-3.83**
msnbc	-6.19	**-0.14**	**-2.62**	**-4.64**
msweb	-9.77	**+0.03**	**-0.07**	**-0.99**
plants	-13.03	**-2.57**	-7.56	-20.92
pumpitup	-14.32	**-0.25**	**-0.25**	**-1.86**
r52	-95.48	**-0.76**	**-1.96**	-6.11
uscensus	-23.20	**-0.27**	**-1.75**	-10.39
andes	-93.23	**-0.49**	-6.22	-16.57
hailfinder	-49.63	**-0.06**	**-2.71**	-10.21
hepar2	-32.60	**-0.28**	**-1.36**	**-3.22**
link	-215.68	**+0.10**	**+1.10**	-16.99
munin1	-41.15	**-0.09**	**-0.16**	-9.95
munin2	-171.82	**-0.02**	**-0.02**	**-1.83**
munin3	-165.09	**0.00**	**0.00**	**-1.10**
munin4	-186.11	**-0.02**	**-0.02**	**-3.86**
pathfinder	-26.65	**-0.66**	**-0.70**	**-4.88**
win95pts	-9.22	**+0.06**	**-1.08**	-9.15

Table 2.5: CVLL score per sample. Every result that is less than 5% smaller than `sota-BNSL`'s score is boldfaced.

dataset	**sota**	**qds**$_{\varepsilon_{0.9}}$ (%)	**qds**$_{\varepsilon_{0.75}}$ (%)	**qds**$_{\varepsilon_{0.5}}$ (%)
20ng	-139.93	**-0.78**	**-2.48**	**-4.92**
adult	-12.73	**-0.42**	**-0.50**	**-4.68**
book	-34.27	**-0.88**	**-1.70**	**-4.79**
covertype	-13.54	**-0.16**	**-1.25**	-12.40
kddcup	-2.37	**-0.30**	**-1.03**	**-3.61**
mnsbc	-6.10	**-0.16**	**-3.54**	**-4.18**
msweb	-8.31	**-0.02**	**-0.11**	**-1.22**
plants	-13.09	**-2.20**	-7.320	-20.85
pumpitup	-14.31	**-0.20**	**-0.20**	**-1.12**
r52	-87.58	**-1.26**	**-2.92**	-7.40
uscensus	-21.77	**-0.41**	**-1.60**	-8.73
andes	-92.87	**-0.53**	-6.30	-17.01
hailfinder	-49.86	**-0.07**	**-2.72**	-10.39
hepar2	-32.64	**-0.31**	**-1.62**	**-3.64**
link	-216.90	**+2.00**	**+1.64**	-10.71
munin1	-37.72	**-0.03**	**-0.06**	-11.74
munin2	-162.89	**-0.02**	**-0.02**	**-2.22**
munin3	-162.59	**+0.00**	**+0.00**	**-1.29**
munin4	-170.50	**+0.03**	**-0.06**	**-4.34**
pathfinder	-21.48	**+0.51**	**-0.07**	**-4.56**
win95pts	-9.48	**+0.00**	**-1.53**	-9.64

Table 2.6: Computation time (seconds). Every result that corresponds to a BDeu score less than 5% smaller than `sota-BNSL`'s score is boldfaced.

dataset	**sota** (*seconds*)	$\mathbf{qds}_{\varepsilon_{0.9}}$ (%)	$\mathbf{qds}_{\varepsilon_{0.75}}$ (%)	$\mathbf{qds}_{\varepsilon_{0.5}}$ (%)
20ng	21,495	**-1.62**	**-42.66**	**-72.94**
adult	1,02	**-6.61**	**-22.03**	**-61.20**
book	7,600	**-23.61**	**-40.33**	**-71.30**
covertype	565	**-6.80** ‘	**-33.22**	-71.13
kddcup	2,167	**-11.49**	**-32.85**	**-73.59**
msnbc	252	**-20.66**	**-60.61**	**-85.65**
msweb	4,701	**-6.29**	**-9.86**	**-55.08**
plants	455	**-46.93**	-61.93	-84.07
pumpitup	6673	**-41.25**	**-41.25**	**-81.82**
r52	18,630	**-13.58**	**-38.47**	-76.71
uscensus	21,782	**-0.44**	**-31.54**	-77.68
andes	898	**-2.23**	-27.42	-69.91
hailfinder	46	**-5.31**	**-17.46**	-54.71
hepar2	76	**-4.05**	**-42.56**	**-70.00**
link	7,240	**-12.03**	**-10.58**	-61.30
munin1	497	**-7.42**	**-17.23**	-59.14
munin2	7,093	**-20.46**	**-21.66**	**-43.68**
munin3	11,558	**-36.91**	**-29.20**	**-54.19**
munin4	8,550	**-7.87**	**-13.08**	**-39.06**
pathfinder	231	**-14.01**	**-35.38**	**-69.48**
win95pts	132	**-6.05**	**-31.41**	-69.07

Table 2.7: Networks' number of arcs. Every result that corresponds to a BDeu score less than 5% smaller than `sota-BNSL`'s score is boldfaced.

dataset	**sota**	$\mathbf{qds}_{\varepsilon\mathbf{0.9}}$ (%)	$\mathbf{qds}_{\varepsilon\mathbf{0.75}}$ (%)	$\mathbf{qds}_{\varepsilon\mathbf{0.5}}$ (%)
20ng	3136	**-4.50**	**-14.89**	**-31.89**
adult	371	**3.23**	**7.01**	**-13.75**
book	2196	**-10.66**	**-19.17**	**-40.30**
covertype	337	**-0.89**	**-11.28**	-37.69
kddcup	285	**-5.26**	**-18.95**	**-38.95**
msnbc	102	**-7.84**	**-33.33**	**-63.73**
msweb	1,264	**-2.53**	**-3.56**	**-34.97**
plants	320	**-6.25**	-18.44	-42.50
pumpitup	46	**-10.87**	**-10.87**	**-15.22**
r52	2713	**-3.65**	**-9.14**	-25.14
uscensus	220	**-10.45**	**-20.45**	-37.73
andes	336	**-0.89**	-7.14	-22.92
hailfinder	64	**-1.56**	**+6.25**	-15.62
hepar2	92	**-3.26**	**-21.74**	**-30.43**
link	1,146	**-1.83**	**-0.44**	-22.43
munin1	208	**0.00**	**+0.96**	-9.62
munin2	879	**0.00**	**0.00**	**-13.31**
munin3	898	**0.00**	**0.00**	**-7.80**
munin4	903	**0.00**	**0.00**	**-8.53**
pathfinder	161	**-4.35**	**-8.70**	**-24.22**
win95pts	115	**0.00**	**-0.87**	-12.17

Score It appears in Table 2.4 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 ($\mathbf{qds}_{\varepsilon_{0.9}}$), and for most of them when 75% of the variables remain ($\mathbf{qds}_{\varepsilon_{0.75}}$). This is also observed with $\varepsilon_{0.5}$ for datasets that contain a lot of very strong pairwise relationships as kddcup, msweb, or munin2-4. Moreover, we remark that for these datasets, the CVLL and BDe scores behave globally the same way. This backs up the fact that BDe was taken as a 'proxy' for generalization capability, and was chosen as the objective in the structure learning search for `sota-BNSL`. Other experiments (not displayed here) show that the BIC score does not lead to the same type of results at all (especially for small data), despite being asymptotically equivalent to BDe.

Computing time Table 2.6 shows a significant decrease in computational time for `qds-BNSL`, which is all the more important that ε 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 $\varepsilon = \varepsilon_{0.5}$ is 55% faster for msweb, and 54% for munin3, while implying only around 1% decrease in score compared to `sota-BNSL`. **If we allow a** 5% **score decrease,** `qds-BNSL` **can be more than** 70% **faster than** `sota-BNSL` (20ng, 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.

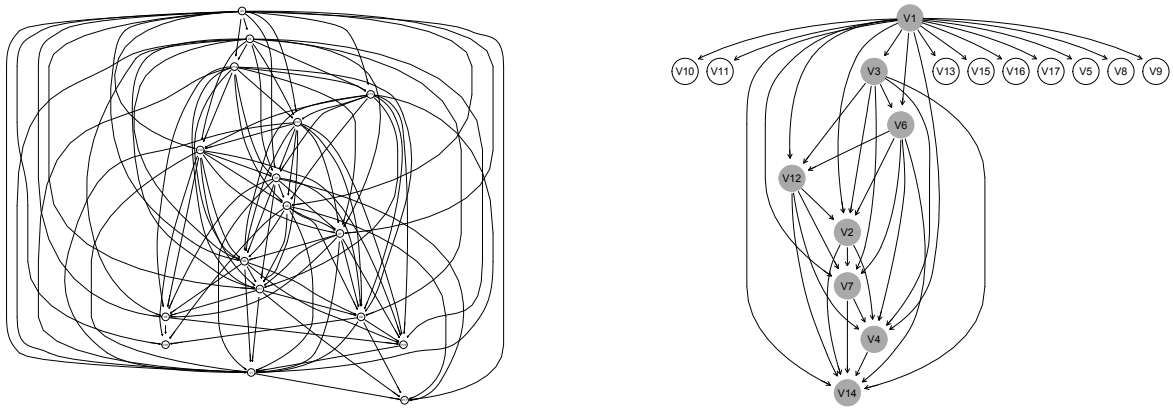
Complexity As showed by Table 2.7, Bayesian networks learned with `qds-BNSL` are consistently less complex than those learned with `sota-BNSL`. Several graphs learned with $\mathbf{qds}_{\varepsilon_{0.5}}$ are more than 30% sparser while still scoring less than 5% below the baseline: 20ng, book, kddcup, msnbc, msweb and hepar2.

Figure 2.5 displays two Bayesian networks learned on the 'msnbc' dataset.

Comparison of computation time, BDeu (normalized), CVLL (normalized), and number of arcs for the displayed Bayesian networks

	Figure 2.5 (a) (sota)	**Figure 2.5 (b) ($\mathbf{qds}_{\varepsilon_{0.5}}$)**
t_{run} (sec)	252	36
BDe score	−6.2	−6.5
CVLL score	−6.1	−6.4
Nb arcs	102	37

They provide an interesting example of the sparsity induced by `qds-BNSL`. After the $\mathbf{qd}_{\varepsilon_{0.5}}$-screening phase, half of the variables (corresponding to the nodes in white) are considered to be



(a) BN returned by `sota-BNSL` **(b)** BN returned by `qds-BNSL`($\varepsilon_{0.5}$)

Figure 2.5: Example of Bayesian networks learned on the `msnbc` dataset

sufficiently explained by variable V_1. 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.

Interpretability 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[15]. The Bayesian network displayed in Figure 2.5(b) shows in a compact way the influence between the different variables. For instance, we see that visits of the website's pages corresponding to nodes in white (*e.g.* 'weather' (V_8), 'health' (V_9) or 'business' (V_{11})) are importantly influenced by whether the user has also visited the frontpage (V_1). For example, learned parameters show that a user who did not visit the website's frontpage or not (V_1) is about 10 times more likely to have visited the website's 'summary' page (V_{13}) than a user who did visit the frontpage. Such information is much harder to read from the graph learned with `sota-BNSL` displayed in Figure 2.5(a), even though its score is a bit higher. Moreover, the low difference in BDe score shows that the generalization performance of the graph is reasonably affected by this simplification.

5.3.4 Results (2/2). Joint consideration of readability / performance and performance / computation time on selected datasets

We have seen in the previous subsection that the quasi-determinism screening approach consistently learns faster, and sparser structures, at the cost of a little generalization performance loss.

[15] more details: `http://archive.ics.uci.edu/ml/machine-learning-databases/msnbc-mld/msnbc.data.html`

Sparser structures are however sometimes more interesting. In the industry notably, interpretable models are more convincing, and we often prefer models that are understandable by domain experts to purely performing models, that are more and more common in the deep learning era. Depending on the final objective, we could therefore accept to sacrifice a bit of performance (in terms of BDe/CVLL score) for sparsity. The same remark goes for computation time, that may be more important than performance in some contexts.

One may then wonder, which method would learn the best performing structure for a given maximum number of arcs, or for a given computation time. In an attempt to tackle this question, we used three methods for learning sparser structures faster, two of which are standard methods, the last one being `qds-BNSL`.

1. Bounding the maximum number of parents per node in the final graph. As we saw earlier in this Chapter, this is a standard method to regularize the structure learning task, and can be used jointly with a score such as MLL to avoid learning overly complex structures. For each of our selected datasets, we ran the `sota-BNSL` algorithm with a restriction on the maximum number of parents per node, ranging from 1 to $n-1$ (no restriction).

2. Reducing the ESS in the baseline hill-climbing algorithm. As thoroughly studied by Silander et al. (2007), the number of arcs of the learned Bayesian network structure is generally highly sensitive with respect to the ESS of the BDe score used in the structure learning optimization problem. We therefore picked a list of ESS ranging from 10^{-16} to 5, and ran our baseline structure learning algorithm `sota-BNSL` using each ESS on that list.

3. Bounding our definition of *quasi determinism* through the ε hyperparameter in the `qds-BNSL` algorithm.

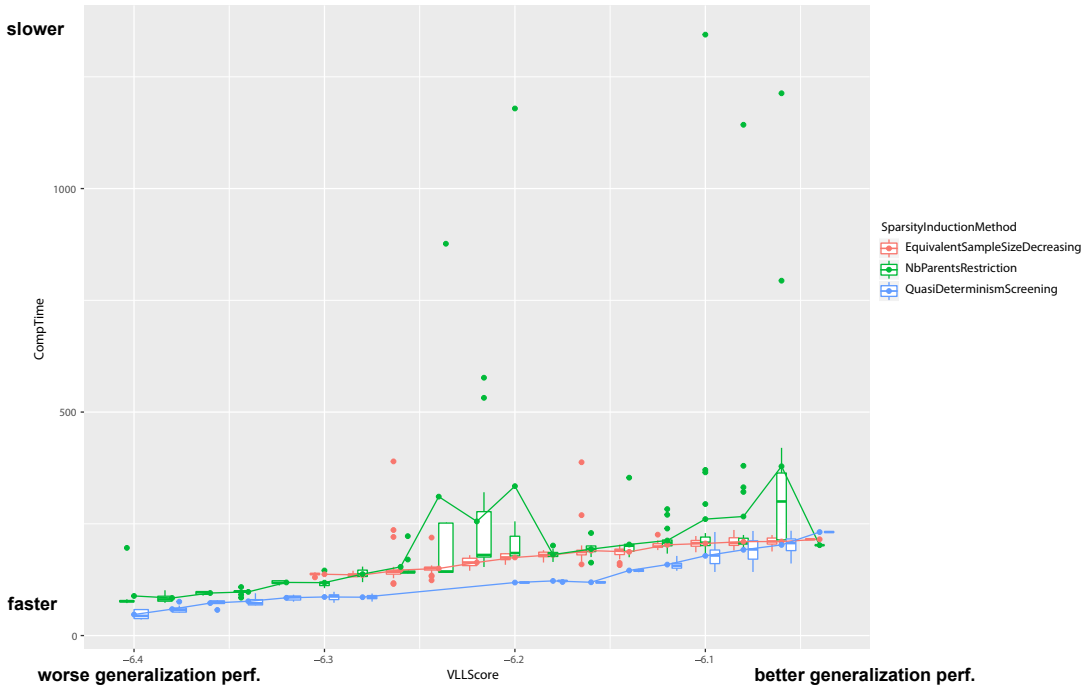
We are interested in looking how `qds-BNSL` performs if we consider it as an algorithm whose primary goal is to learn sparser structures faster. Figures 2.6, 2.7 and 2.8 present the results on 3 selected datasets (representative of the spectrum of efficiency of `qds-BNSL`), by displaying the characteristics of the structures learned with the 3 different structure learning algorithms on *Performance* $\times$ *Readability* and *Computation Time* $\times$ *Performance* axis.

These figures suggest that **the performance / computation time trade-off is always in the advantage of the `qds-BNSL` algorithm**. In other terms, given a fixed BDe score, `qds-BNSL` is quicker to learn a graph with such a score than `sota-BNSL`.

As for the **readability / performance trade-off**, *i.e.* looking at which algorithm learns the best scoring graph with a given number of arcs, **`qds-BNSL` has better results on two out of three datasets: `pumpitup` and `msnbc`**, as seen in Figures 2.8 and 2.6. We see in Figure 2.7

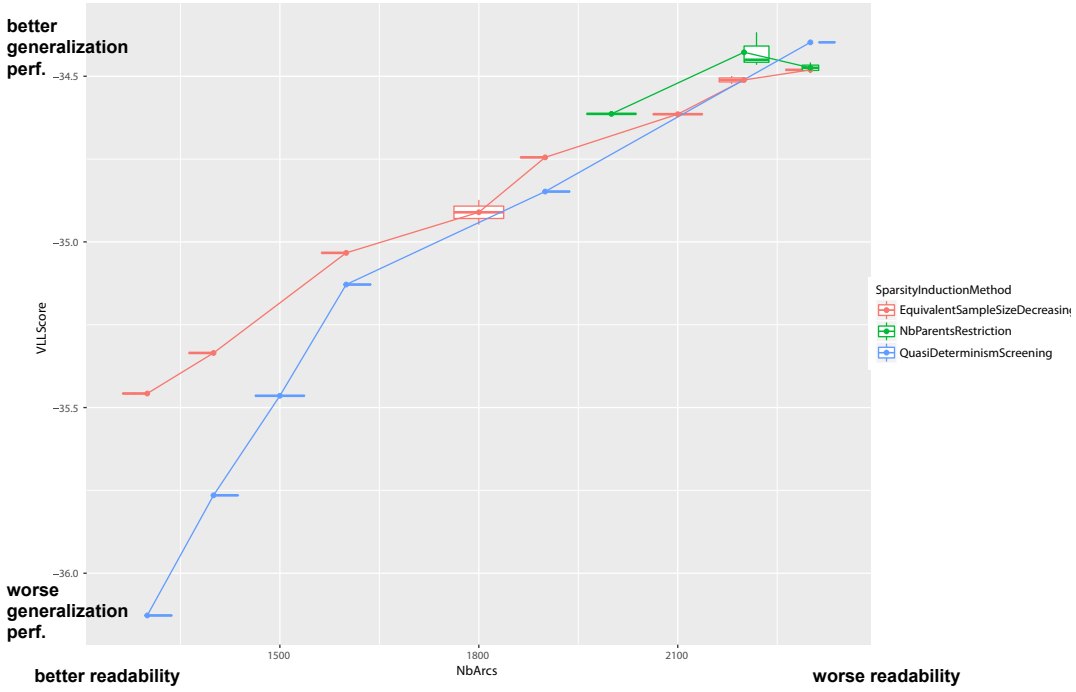
that in the case of the `book` dataset, the sparsity induced by decreasing the ESS of the target BDe score affects less the CVLL score. This dataset was chosen on purpose as it shows a case where the `qds-BNSL` is not at 'its best'. This highlights the fact that the potential of the `qds-BNSL` algorithm and the optimal way to choose ε when running it, are very **data-dependent** properties. As we will talk about later (Chapter 4), this remark motivates one of the most important research perspective to this work.

(a) performance/readability tradeoff - `msnbc` dataset

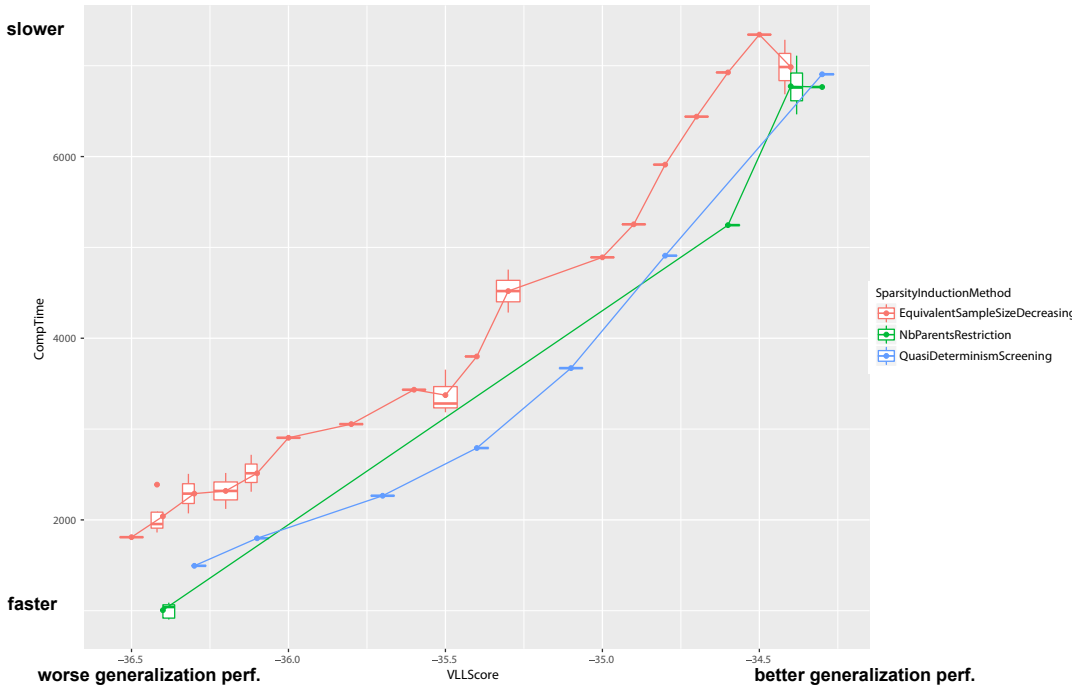


(b) performance/computation time tradeoff - `msnbc` dataset

Figure 2.6: Graphical representation of performance trade-offs for the *msnbc* dataset

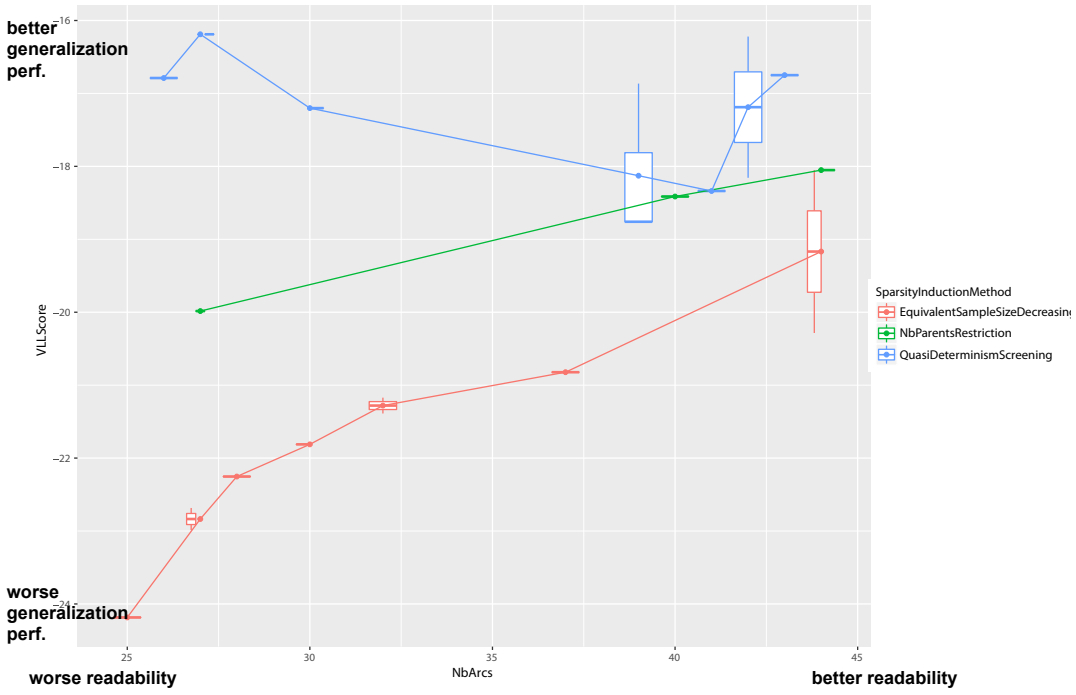


(a) performance/readability tradeoff - `book` dataset

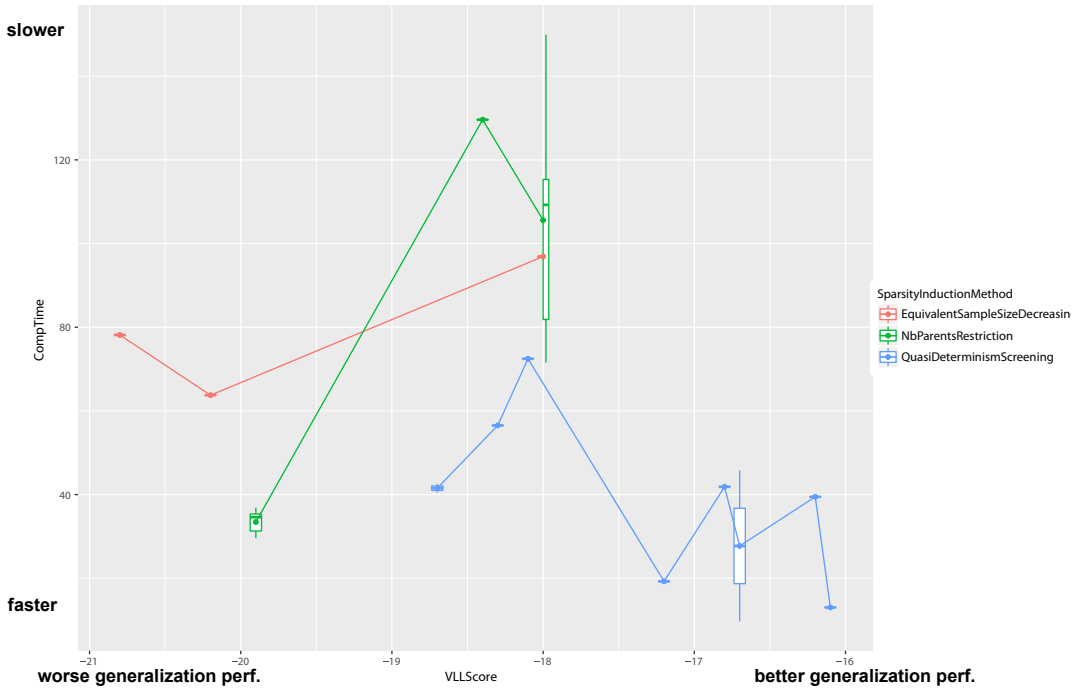


(b) performance/computation Time tradeoff - `book` dataset

Figure 2.7: Graphical representation of performance trade-offs for the *book* dataset



(a) performance/readability tradeoff - `pumpitup` dataset



(b) performance/computation Time tradeoff - `pumpitup` dataset

Figure 2.8: Graphical representation of performance trade-offs for the *pump it up* dataset

6 Concluding remarks

6.1 Summary

We have seen that, both in theory and in practice, the (quasi-)determinism screening approach enables a significant decrease in computational time and complexity for a small decrease in graph score.

Deterministic screening is **consistent w.r.t. the MLL score**, and datasets that contain an important number of deterministic relationships by design are learned better and faster with the `ds-BNSL` algorithm compared to a standard baseline `sota-BNSL`.

When we are facing standard data however, theoretical guarantees do not hold, and the learned Bayesian Network `qds-BNSL` are often a bit less performing than those learned with `sota-BNSL`. However, they often have very interesting **performance-vs-readability**, *i.e.* they are more performing than graphs with the same arc density learned with regularized versions of `sota-BNSL`, as well as consistent **computation time-vs-performance** tradeoffs, *i.e.* they are **faster** to compute for a given performance score than with `sota-BNSL`.
However these properties highly depend on the dataset. These tradeoffs are all the more advantageous as 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 baseline structure learning algorithm during the second phase of `qds-BNSL`.
Optimal cases for this algorithm take place when $n_r(\varepsilon)$ is significantly smaller than n for ε reasonably small compared to the variables entropies. In practice this is reasonably frequent (*e.g* 20 newsgroup, msnbc, munin2-4, webkb among others).

Note that in any case, the speed of the `qds-BNSL` algorithm, and the fact that it is good at highlighting very strong pairwise relationships, makes it a very interesting tool for dataset exploration during a pre-processing phase, even in cases where it is not used to learn the final model.

6.2 Some perspectives

As we saw in the previous result section, the (quasi-)determinism screening approach is indubitably better in some cases, the most obvious of which being in presence of determinism. However, its impact is not as great in other cases, even if it consistently decreases computation time and increases the structures' sparsity.

Our main research perspective is to be able to **anticipate how good the tradeoff may be** before running any algorithm all the way through, saving us from running `qds-BNSL` on

datasets in which there are absolutely no strong pairwise relationships, and enabling us to choose an optimal value of ε on datasets on which `qds-BNSL` has potential, *i.e.* the value of ε that introduces the most sparsity and time gain for the smallest graph 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 ε for a given dataset.
In Chapter 4, we will present ideas and some preliminary results, both concerning guarantees for scores other than the MLL score, and concerning ways to choose the value of the ε in the `qds-BNSL` algorithm.

Besides, we still have potential to improve our current implementation of the `qds-BNSL` algorithm, by parallelizing the computation of $\mathbb{H}^D$, and implementing it in `C` instead of `R`.

Finally, we have some insights on ways to generalize our quasi-determinism screening idea. The proof of Proposition 4 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 dependence discovery methods.
We also could use **alternative definitions for quasi-determinism**: instead of considering the information-theoretic quantity $H^D(X|Y)$ to describe the strength of the relationship $Y \to X$, one could choose $\frac{H^D(X|Y)}{H^D(X)}$, which represents the proportion of X's entropy that is explained by Y. This would allow us to express ε as a percentage (with a uniform scale across datasets). Moreover, $\frac{H^D(X|Y)}{H^D(X)} \leq \varepsilon$ can be rewritten as $\frac{I^D(X,Y)}{H(X)} \geq 1 - \varepsilon$, which gives another insight to quasi-determinism screening: for a given variable X, this comes down to finding a variable Y such that $I^D(X, Y)$ 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 $\varepsilon-$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).

Chapter 3

Bayesian networks for joint modeling of temporal and static data

Contents

In this chapter, we first present a model to address the objective of static and temporal data fusion announced in the Introduction. We refer to this model as the *hybrid static-dynamic Bayesian network* (Section 1). Associated algorithms for using such a model for inference tasks, as well as learning it from data are then detailed (Section 2). In Section 3, we explain how the theoretical framework described in Section 1 can be obtained from the data that is available in practice from an evolving system, such as Schneider Electric's connected products.

1 Hybrid static-dynamic Bayesian networks for static and temporal data fusion: overview

We consider a setting of static and temporal data, in which the static data correspond to metadata describing each time series which are observed in the temporal data. Once this formal setting is introduced, we present the hybrid static-dynamic Bayesian network, which jointly models static and temporal variables. Throughout the section, we use an example to illustrate both the setting and the model.

1.1 Theoretical framework

1.1.1 Static and temporal variables

We suppose $\mathbf{X} = (X_1, \dots, X_n)$ is a tuple of categorical static variables (descriptive metadata), and $\mathbf{Y} = (Y_1, \dots Y_k)$ is a $k-$tuple of continuous variables such that $\forall \mathbf{x} \in Val(\mathbf{X})$, $\mathbf{Y}|\mathbf{X} = \mathbf{x}$ corresponds to a (multivariate) time series observed on a set of successive time stamps (independent from $\mathbf{x}$) $\{t_1, \dots, t_l\}$, where $l \in \mathbb{N} \setminus \{0\}$. We assume that the time step is constant: $\exists \Delta t > 0$ such that $\forall j \in [\![1, l-1]\!]$, $t_{j+1} - t_j = \Delta t$.

For $\mathbf{x} \in Val(\mathbf{X})$, we use the notation:

$$\mathbf{Y}_{\mathbf{x}} = (Y_{\mathbf{x},1}, \dots, Y_{\mathbf{x},k}) \tag{3.1}$$

to denote the temporal variable with distribution $P(\mathbf{Y}|\mathbf{X} = \mathbf{x})$. The associated multivariate time series is:

$$\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}.$$

Moreover, we make the following assumptions:

Assumption (A) *For each $\mathbf{x} \in Val(\mathbf{X})$, $\mathbf{Y}_{\mathbf{x}}$ satisfies Assumptions 1 to 4 mentioned in Section 2.1.4 of Chapter 1. In other words, each time series $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}$ can be modeled with a 2-time-step Bayesian network (2TBN) presented in Section 2.3.1 of Chapter 1.*

Assumption (B) *The structure of the DBN modeling* $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1\leq j\leq l}$ *does not depend on* $\mathbf{x} \in Val(\mathbf{X})$.

The second hypothesis may seem restrictive, but we believe it to be reasonable in practice, as we illustrate through examples later on.

Lastly, $\tilde{\mathbf{Y}}$ denotes the Δ_t-**shifted** version of variable $\mathbf{Y}$, *i.e.* $\forall j \in [\![1, l-1]\!]$, and $\forall \mathbf{x} \in Val(\mathbf{X})$,

$$\begin{aligned}\tilde{\mathbf{Y}}_{\mathbf{x}}(t_j) &= \mathbf{Y}_{\mathbf{x}}(t_j + \Delta_t) \\ &= \mathbf{Y}_{\mathbf{x}}(t_{j+1}).\end{aligned}$$

1.1.2 Datasets

We suppose that $D^{\mathbf{X}}$ is a dataset containing M observations of $\mathbf{X}$, and satisfying the metadataset assumption, *i.e.*:

Metadataset assumption *The rows of* $D^{\mathbf{X}}$ *are distinct, i.e.* $\forall m_1, m_2 \in [\![1, M]\!]$,

$$\mathbf{x}^{(m_1)} = \mathbf{x}^{(m_2)} \Rightarrow m_1 = m_2.$$

This is indeed typically verified in metadatasets, since every row of such a set identifies entirely a time series, thus preventing the existence of identical rows.

Note that if $D^{\mathbf{X}}$ satisfies the metadataset assumption, and if F is a deterministic forest w.r.t. $D^{\mathbf{X}}$, then the observations of variables $\mathbf{X}_{\mathcal{R}(\mathcal{F})}$ are sufficient to entirely identify the rows of $D^{\mathbf{X}}$, *i.e.* $\forall m_1, m_2 \in [\![1, M]\!]$,

$$\mathbf{x}_{\mathcal{R}(\mathcal{F})}^{(m_1)} = \mathbf{x}_{\mathcal{R}(\mathcal{F})}^{(m_2)} \Rightarrow m_1 = m_2.$$

In practice, there often are a small number of variables $\mathbf{X}_{\mathcal{R}(\mathcal{F})}$, corresponding to the different descriptive *dimensions* of metadata: **measuring device**, **measured quantity**, **location**, **associated asset**, *etc.* Note that these correspond to the non-temporal branches of the underlying **star / snowflake schema**[1].
Figure 3.1 displays an example of metadata organized as dimensions according to a snowflake schema.

We suppose that we observe **one** multivariate time series $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1\leq j\leq l}$ per value $\mathbf{x} \in Val(\mathbf{X})$[2]. We will denote by $D^{\mathbf{XY}}$ the dataset containing observations of $\mathbf{Y}_{\mathbf{x}}(t)$ and its associated metadata $\mathbf{x}$, for all time stamps $t \in \{t_1, \ldots, t_l\}$ and all accessible metadata configurations $\mathbf{x} \in Val(\mathbf{X})$.

[1] In the business intelligence domain, the star and snowflakes schemas (Ralph, 1996) are used to efficiently query temporal data based on static metadata.

[2] Here $Val(\mathbf{X})$ contains every value of $\mathbf{X}$ that is observed in the metadata table $D^{\mathbf{X}}$. We should expect $|Val(\mathbf{X})| = M$ to be significantly smaller than $\prod_{i=1}^{n} |Val(X_i)|$, because of redundancy and determinism.

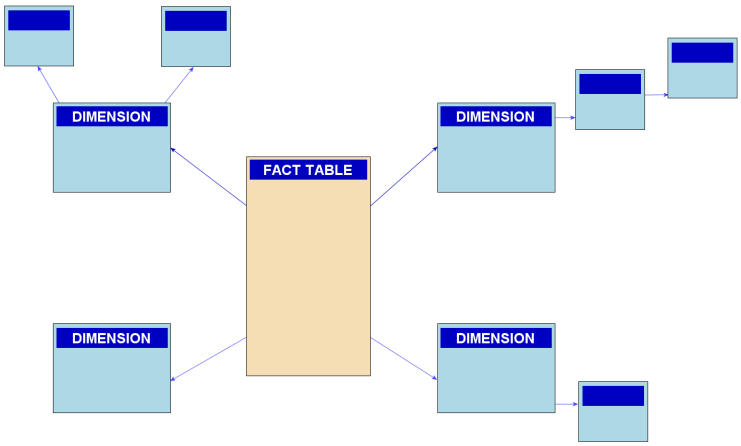


Figure 3.1: Example of a snowflake schema

$D^{\mathbf{XY}}$ therefore contains $l \times |Val(\mathbf{X})|$ rows and $n+k$ columns (not counting the time stamp). Each row corresponds to a given configuration $\mathbf{x}$ of $\mathbf{X}$ and a given time step t_j for $j \in [\![1, l]\!]$, and is formed of the $n+k$ observations of variables $\mathbf{X}$ and $\mathbf{Y}(t_j)$: $\{x_1, \ldots, x_n, y_{\mathbf{x},1}(t_j), \ldots, y_{\mathbf{x},k}(t_j)\}$.

Finally, $D^{\mathbf{XY\tilde{Y}}}$ denotes the dataset derived from $D^{\mathbf{XY}}$, in which we also add the observations of the shifted time series $\tilde{\mathbf{Y}}_{\mathbf{x}}$ for all $\mathbf{x} \in Val(\mathbf{X})$.
$D^{\mathbf{XY\tilde{Y}}}$ has $(l-1) \times |Val(\mathbf{X})|$ rows and $n+2k$ columns (not counting the time stamp). Each row corresponds to a given time step t_j for $j \in [\![1, l-1]\!]$ and a given configuration $\mathbf{x}$ of $\mathbf{X}$, and is formed of the $n+2k$ observations of variables $\mathbf{X}$, $\mathbf{Y}(t_j)$ and $\tilde{\mathbf{Y}}(t_j) = \mathbf{Y}(t_{j+1})$: $\{x_1, \ldots, x_n, y_{\mathbf{x},1}(t_j), \ldots, y_{\mathbf{x},k}(t_j), y_{\mathbf{x},1}(t_{j+1}), \ldots, y_{\mathbf{x},k}(t_{j+1})\}$.

This setting is naturally derived, through a pre-processing phase, from the data that is accessible in practice from an IoT system, as we explain in Section 3. We now present a simple example to illustrate the notations we introduced.

1.1.3 Example ($\star$): presentation of variables and associated datasets

We introduce an example that will be used throughout this Chapter. The joint modeling of temporal and static data implies technical notations, that are not easy to grasp outside of a concrete case.

In example ($\star$), we consider a small part of an office building, containing six zones (given by variable X_1) spread upon 3 rooms (given by variable X_2).
Table 3.1 represents the associated metadataset $D^{\mathbf{X}}$.

This dataset satisfies the *metadataset assumption*. In this case, the `Zone` variable X_1 actually contains all the information: there exists only one row per given value of X_1.
$Val(\mathbf{X})$ = { (Perch1, OpenSpace), (Perch2, OpenSpace), (Perch3, OpenSpace), (MeetingRoomEast,

X_1 (Zone)	X_2 (Room)
Perch1	OpenSpace
Perch2	OpenSpace
Perch3	OpenSpace
MeetRoomEast	MeetingRoom
MeetRoomWest	MeetingRoom
Box	Box

Table 3.1: Metadataset $D^{\mathbf{X}}$ in the case of example ($\star$)

MeetingRoom), (MeetingRoomWest, MeetingRoom), (Box, Box) } contains 6 elements: one per row of $D^{\mathbf{X}}$. Finally, we remark that the relationship $X_1 \Rightarrow X_2$ is deterministic.

We suppose that we observe a multivariate temporal variable $\mathbf{Y} = (Y_1, Y_2, Y_3)$, corresponding to the three following quantities:

- Y_1: level of CO_2 (*ppm*),
- Y_2: temperature (*degree Celsius*),
- Y_3: number of people.

We suppose that each temporal variable is observed at a collection of time stamps $\{t_1, \ldots, t_l\}$, which is the same for all $\mathbf{x} \in Val(\mathbf{X})$. Furthermore, we suppose that $\forall j \in [\![1, l]\!]$, $t_{j+1} - t_j$ is constant (typically equal to 15min).

For each $\mathbf{x} \in Val(\mathbf{X})$, $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}$ is the (multivariate) time series corresponding to the observation of these three quantities in the area of the building identified by $\mathbf{x}$.

Table 3.2 represents an extract of the associated dataset $D^{\mathbf{XY}}$.

Finally, Table 3.3 represents an extract of $D^{\mathbf{XY\tilde{Y}}}$.

Note that **assumption (B)** proposed in Section 1.1.1 makes sense in the setting of example ($\star$). Indeed, the structure of the DBN modeling $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}$ can be intuitively expected not to depend on $\mathbf{x} \in Val(\mathbf{X})$, *i.e.* the (in)dependencies in between the physical quantities through time should realistically not depend on the zone or the room (*e.g.* a high number of people has a positive impact on CO2 and temperature, whatever the considered area). However, we can expect the quantitative information underlying these relations (encoded by the parameters of the network) may vary, *e.g.* the temperature increase rate when people are present in a room should depend in the size of said room.

	X_1 (zone)	X_2 (room)	Y_1 (CO_2 level)	Y_2 (temperature)	Y_3 (nb people)
t_1	Perch1	OpenSpace	3.2	21.3	2
t_2	Perch1	OpenSpace	3.3	21.1	3
...	...	...	...	...	...
t_{l-1}	Perch1	OpenSpace	4.9	19.4	1
t_l	Perch1	OpenSpace	4.7	19.8	2
...	...	...	...	...	...
t_1	MeetRoomEast	MeetingRoom	3.0	20.7	0
t_2	MeetRoomEast	MeetingRoom	2.3	21.0	8
...	...	...	...	...	...
t_{l-1}	MeetRoomEast	MeetingRoom	5.1	19.9	10
t_l	MeetRoomEast	MeetingRoom	5.5	20.2	11
...	...	...	...	...	...

Table 3.2: Extract of the dataset $D^{\mathbf{XY}}$ in the case of example ($\star$)

	X_1 (zone)	X_2 (room)	Y_1 (CO_2 level)	Y_2 (temperature)	Y_3 (nb people)	$\tilde{Y}_1$ (CO_2 level)	$\tilde{Y}_2$ (temperature)	$\tilde{Y}_3$ (nb people)
t_1	Perch1	OpenSpace	3.2	21.3	2.0	3.3	21.1	3.0
...	...	...	...	...	...	...	...	...
t_{l-1}	Perch1	OpenSpace	4.9	19.4	1.0	4.7	19.8	2.0
...	...	...	...	...	...	...	...	...
t_1	MeetRoomEast	MeetingRoom	3.0	20.7	0	2.3	21.0	8
...	...	...	...	...	...	...	...	...
t_{l-1}	MeetRoomEast	MeetingRoom	5.1	19.9	10	5.5	20.2	11
...	...	...	...	...	...	...	...	...

Table 3.3: Extract of the dataset $D^{\mathbf{XY}\tilde{\mathbf{Y}}}$ in the case of example ($\star$)

1.2 Hybrid static-dynamic Bayesian network

1.2.1 Model description

In this subsection, we propose the **hybrid static-dynamic Bayesian network** (HSDBN) model, using ideas both from static Bayesian networks and dynamic Bayesian networks, in order to jointly model temporal and static data. As explained in the introduction, we believe that this will notably enable to automate existing tasks that currently need expert knowledge input because of the fact that static (metadata) is not included as such in the model.

Notations We denote by $V^{\mathbf{X}} = \{v_1^{\mathbf{X}}, \ldots, v_n^{\mathbf{X}}\}$ the set of nodes associated with the (categorical) static variables $\mathbf{X} = (X_1, \ldots, X_n)$.
We denote by $V^{\mathbf{Y}} = \{v_1^{\mathbf{Y}}, \ldots, v_k^{\mathbf{Y}}\}$ and $V^{\tilde{\mathbf{Y}}} = \{v_1^{\tilde{\mathbf{Y}}}, \ldots, v_k^{\tilde{\mathbf{Y}}}\}$ the sets of nodes, respectively associated with the (continuous) temporal variables $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$.

We do not refer to nodes of the graph G by integers as in Chapter 2, since there now are different categories of variables.

Model overview We choose to model the static and temporal variables with a particular case of Bayesian network, that we call *hybrid dynamic-static*.
This Bayesian network formally models $(\mathbf{X}, \mathbf{Y}, \tilde{\mathbf{Y}})$, that we consider to be simultaneously observable variables (as in $D^{\mathbf{XY}\tilde{\mathbf{Y}}}$), and is defined as:

$$\mathcal{B} = (G, \Theta)$$

where:

- $G = (V, A)$ is a DAG with:
  - $V = V^{\mathbf{X}} \cup V^{\mathbf{Y}} \cup V^{\tilde{\mathbf{Y}}}$,
  - $A \subset V^2$ satisfying the following structural constraints: for any $(v_1, v_2) \in A$, we must have:
    - $v_1 \notin V^{\tilde{\mathbf{Y}}}$,
    - $v_1 \in V^{\mathbf{Y}} \Rightarrow v_2 \in V^{\tilde{\mathbf{Y}}}$
- $\Theta = \{\Theta_i^{\mathbf{X}}\}_{1 \leq i \leq n} \cup \{\Theta_i^{\mathbf{Y}}\}_{1 \leq i \leq k} \cup \{\Theta_i^{\tilde{\mathbf{Y}}}\}_{1 \leq i \leq k}$ with each parameter Θ_i defining the local distribution of its associated variable given its parent variables in G.

Parent functions in structure G The parent function $\pi^G(.)$ presented in Section 2.1.1 of Chapter 1 needs to be adapted to the new node notation.
We note that the structural constraints on G imply that:

- only nodes in $V^{\tilde{\mathbf{Y}}}$ can have parents in $V^{\mathbf{Y}}$,
- all of the nodes in V can have parents in $V^{\mathbf{X}}$.

Therefore, we define the new **parent functions** $\pi^G_{\mathbf{Y}}$ and $\pi^G_{\mathbf{X}}$, that return the parent indexes among variables $V^{\mathbf{Y}}$ and $V^{\mathbf{X}}$ respectively: for $G = (V, A)$ a HSDBN structure in accordance with the previously defined notations,

$$\pi^G_{\mathbf{Y}} : \left| \begin{array}{ccl} V^{\tilde{\mathbf{Y}}} & \longrightarrow & 2^{[\![1,k]\!]} \\ v & \longmapsto & \{j \in [\![1,k]\!] \mid (v_j^{\mathbf{Y}}, v) \in A\}, \end{array} \right.$$

and

$$\pi^G_{\mathbf{X}} : \left| \begin{array}{ccl} V & \longrightarrow & 2^{[\![1,n]\!]} \\ v & \longmapsto & \{i \in [\![1,n]\!] \mid (v_i^{\mathbf{X}}, v) \in A\}. \end{array} \right.$$

In short, $\pi^G_{\mathbf{Y}}$ can be used for arcs $\mathbf{Y} \to \tilde{\mathbf{Y}}$, and $\pi^G_{\mathbf{X}}$ for arcs $\mathbf{X} \to \mathbf{X}$, $\mathbf{X} \to \mathbf{Y}$ and $\mathbf{X} \to \tilde{\mathbf{Y}}$.

HSDBN structure: link with previous assumptions

- **Assumptions (A1-A2-A3)** (Assumptions 1,2,3 stated in Section 2.1.4 of Chapter 1) imply that it is legitimate to consider $\mathbf{Y}$ only on short intervals of two successive time stamps, to model its evolution through time. It also guarantees that valid arcs concerning nodes in $V^{\mathbf{Y}} \cup V^{\tilde{\mathbf{Y}}}$ can only go from nodes in $V^{\mathbf{Y}}$ to nodes in $V^{\tilde{\mathbf{Y}}}$, hence our structural constraints do not prevent valid arcs to appear.

- **Assumption (A4)** (Assumptions 4 stated in Section 2.1.4 of Chapter 1) states that the structure representing the dependencies between $\mathbf{Y}$ observed at two successive time stamps does not depend on the time stamp. **Assumption (B)** moreover implies that this structure should not depend on the configuration $\mathbf{x}$ of $\mathbf{X}$ neither. This justifies that we have a single DBN structure representing the evolution of $\mathbf{Y}_{\mathbf{x}}$ through time for all $\mathbf{x}$: the value $\mathbf{x}$ of $\mathbf{X}$ does not affect the structure in between nodes in $V^{\mathbf{Y}}$ and $V^{\tilde{\mathbf{Y}}}$, it does however affect the corresponding parameters.

HSDBN parameters interpretation The constraints on the model structure imply that:

- Nodes in $V^{\mathbf{X}}$ correspond to categorical variables (that are all static). They can only have nodes in $V^{\mathbf{X}}$ as parents. The corresponding parameters are simply conditional probability tables as explained in Chapter 1.

- Nodes in $V^{\mathbf{Y}}$ correspond to continuous variables. They can only have nodes in $V^{\mathbf{X}}$ as parents. The distribution of a variable Y_i for $i \in [\![1, k]\!]$, is therefore modeled as a mixture of Gaussian distributions, with as many components as there are configurations of its parent variables $\mathbf{X}_{\pi^G_{\mathbf{X}}(v_i^{\mathbf{Y}})}$.

- Nodes in $V^{\tilde{\mathbf{Y}}}$ correspond to continuous variables. They can have both nodes in $V^{\mathbf{Y}}$ and nodes in $V^{\mathbf{X}}$ as parents. The distribution of a variable $\tilde{Y}_i$ for $i \in [\![1, k]\!]$, is therefore given by a mixture of Gaussian distributions, with as many components as there are configurations of its categorical parent variables $\mathbf{X}_{\pi^G_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}$.
  For each configuration $\mathbf{x}_{\pi^G_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}$ of $\mathbf{X}_{\pi^G_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}$, the mean of the corresponding Gaussian distribution depends linearly on the values of its continuous parent variables $\mathbf{Y}_{\pi^G_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})}$, and the standard deviation is fixed (depends only on the configuration $\mathbf{x}_{\pi^G_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}$).

  In simpler terms, this means that the 'dynamic' part of the model contains by design a different set of parameters for each time series $\{\mathbf{Y}_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}$ for $\mathbf{x} \in Val(\mathbf{X})$, depending on the parents among $V^{\mathbf{X}}$ of nodes $V^{\tilde{\mathbf{Y}}}$. In the most extreme case, we have a different set of parameter for every $\mathbf{x} \in Val(\mathbf{X})$ (but the same DBN structure).

1.2.2 Example ($\star$) continued: hybrid static-dynamic Bayesian network

We consider the setting described in Section 1.1.3. The graph displayed in Figure 3.2 is an example of HSDBN jointly modeling the static variables $\mathbf{X}$ and the temporal variables $\mathbf{Y}$.

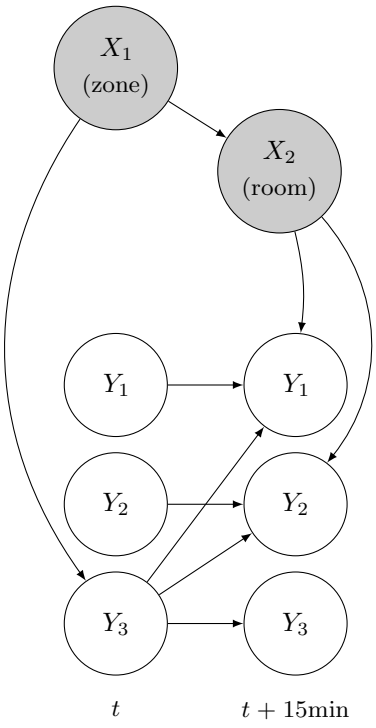


Figure 3.2: Example structure of a hybrid static dynamic Bayesian network in the setting of example ($\star$). Static nodes are colored in light gray and temporal nodes in white.

Here are some insights on the interpretation of this structure:

- The parents of Y_1 (at $t + 15\text{min}$) are: Y_1, Y_3 (at t) and X_2.
  The evolution of the level of CO_2 (Y_1) is therefore modeled as depending on the number of people in the room (Y_3) and the level of CO_2 (Y_1) at the previous time stamp (15 minutes before). This dependency is modeled as linear, and the corresponding set of coefficients is different for every room (X_2), which seems intuitive: the evolution of the level of CO_2 relative to the number of people should depend highly on the considered room.

- The evolution of the temperature (Y_2) can be interpreted in a similar way.

- The only parent of Y_3 (at $t + 15\text{min}$) is Y_3 (at t).
  The evolution of the number of people in the room (Y_3) is therefore modeled as depending on the number of people in the corresponding room at the previous time stamp only[3], and

[3] Y_3 represents the number of people, and is therefore not a continuous variable by design. In practice however, the values of Y_3 are often averages over a given time bin (15min here) and are therefore not consistently integers. We make the choice to model this variable as a conditional Gaussian, like the other temporal variables.

does not depend on the localization (zone or room).

However Y_3 (at t) has X_1 as a parent.

The number of people in a given zone *a priori* (without any information concerning the number of people in the past) is therefore modeled as different for every zone (X_1).

Table 3.4 presents an extract of possible parameters associated with the structure presented in Figure 3.2.

x_2	μ_{x_2}	σ_{x_2}
Perch1	1.2	0.4
Perch2	2.5	1.0
Perch3	1.1	0.4
MeetRoomEast	2.7	1.4
MeetRoomWest	1.1	0.6
Box	0.6	0.3

(a) $\Theta_3^Y = \{(\mu_{x_2}, \sigma_{x_2})\}_{x_2 \in Val(X_2)}$, giving the parameters of the distribution $Y_3|X_2 = x_2 \sim \mathcal{N}(\mu_{x_2}, \sigma^2_{x_2})$

x_1	$\boldsymbol{\beta}_{x_1}$	σ_{x_1}
OpenSpace	$(0.4, 0.9, 0.1)$	0.3
Box	$(3.1, 0.8, 0.7)$	0.6
MeetingRoom	$(-0.1, 0.9, 0.3)$	0.5

(b) $\Theta_2^{\tilde{Y}} = \{(\boldsymbol{\beta}_{x_1}, \sigma_{x_1})\}_{x_1 \in Val(X_1)}$, giving the parameters of the distribution $\tilde{Y}_2|\{X_1 = x_1, Y_2 = y_2, Y_3 = y_3\} \sim \mathcal{N}(\mu_{x_1}(y_2, y_3), \sigma^2_{x_1})$, where $\mu_{x_1}(y_2, y_3) = (1, y_2, y_3)\boldsymbol{\beta}^T_{x_1}$

Table 3.4: Example parameters for distributions $Y_3|X_2$ and $\tilde{Y}_1|X_1, Y_1, Y_3$, consistent with the structure presented in Figure 3.2

These parameters can be interpreted in a straightforward way:
For example, the number of people in a given area (Y_3) (without knowledge on past values) is modeled as a Gaussian variable with a mean that depends only on the considered zone (X_2). This makes sense as all the zones in the open space may not contain the same number of seats, and people may tend to seat more on one side of the meeting room than the other (especially when there are few of them).
We also observe that the number of people (Y_3) has a positive influence on the future temperature ($\tilde{Y}_2$), and that this effect is more important in a small room (*i.e.* Box) than in the open space.

Translating the conditional distribution in intuitive terms, we can write the following relations:

In any zone of the open space,

$$\texttt{Temperature}(t + 15min) = 0.4 \ + \ 0.9 \times \texttt{Temperature}(t) + 0.1 \times \texttt{NumberPeople}(t) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, 0.3^2)$.

In the box,

$$\texttt{Temperature}(t + 15min) = 3.1 \ + \ 0.8 \times \texttt{Temperature}(t) + 0.7 \times \texttt{NumberPeople}(t) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, 0.6^2)$.

In the Perch3 zone, we have the following model of the number of people a priori at a given time t, *i.e.* in the absence of information on the past,

$$\texttt{NumberPeople}(t) = 1.1 \ + \ \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, 0.4^2)$.

Intuitively, we see some additional information could be brought to this model. For example, categorical time information such as the *day of the week*, or the *time of day* could be quite helpful to model the number of people with greater accuracy. This will be discussed in Section 3.3.

2 Inference and learning algorithms for hybrid static-dynamic Bayesian networks

In this section we propose (i) an **inference algorithm** adapted to HSDBN models to perform metadata recovery from a sequence of temporal observations, and (ii) a **learning algorithm** for HSDBN models, relying both on the `ds-BNSL` algorithm presented in Chapter 2 and on standard structure learning methods.

2.1 Inference

Many queries that are not studied in this section can be straightforwardly answered using standard inference algorithms for Bayesian networks, without any adaptation to our hybrid static and dynamic context required.
In this section, we will focus on a specific inference task that implies an adaptation to the hybrid static and dynamic context, and that represents one of the most current setting we face in reality: metadata recovery from a sequence of temporal observations.

Here, we suppose we are in the setting described in Section 1.1: we consider conjointly a tuple of n static descriptive variables $\mathbf{X}$ and a tuple of k temporal variables $\mathbf{Y}$. Moreover, we suppose $\mathcal{B} = (G, \Theta)$ is a hybrid static-dynamic Bayesian network modeling jointly the distributions of $\mathbf{X}$ and $\mathbf{Y}$, as described in Section 1.2.

2.1.1 Metadata recovery from a sequence of temporal observations

We suppose that we observe a sequence

$$\{\mathbf{y}_{\mathbf{x}}(t_1), \ldots, \mathbf{y}_{\mathbf{x}}(t_d)\}$$

of $d \in \mathbb{N}^*$ successive realizations of a temporal variable $\mathbf{Y}_\mathbf{x}$ corresponding to an unknown (fixed) configuration $\mathbf{x} \in Val(\mathbf{X})$, on time stamps $(t_1, \ldots, t_d)$[4].

Metadata recovery from a sequence of temporal observations formally comes down to the estimation of the distribution $P(\mathbf{X}|\mathbf{E})$, where:

$$\mathbf{E} = \{\mathbf{Y}(t_1) = \mathbf{y}^{(1)}, \ldots, \mathbf{Y}(t_d) = \mathbf{y}^{(d)}\}, \tag{3.2}$$

and under the assumption that $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)} \in Val(\mathbf{Y})$ are successive realizations of a single time series $\mathbf{Y}_\mathbf{x}$, corresponding to a fixed value $\mathbf{x}$ of $\mathbf{X}$.

In the context of metadata recovery, we sometimes are not interested in the estimation of the whole distribution $P(\mathbf{X}|\mathbf{E})$, but only in its mode $\hat{\mathbf{x}}^{MAP}$ defined as:

$$\hat{\mathbf{x}}^{MAP} \in \underset{\mathbf{x} \in Val(\mathbf{X})}{\operatorname{argmax}} \; P(\mathbf{X} = \mathbf{x}|\mathbf{E}). \tag{3.3}$$

This may be compared to the general problem of **model identification**, where $\mathbf{x}$ plays the role of a fixed underlying model (the parameters of which are encoded into $\mathcal{B}$), representing the evolution of the temporal variable $\mathbf{Y}$: for a given configuration $\mathbf{x}$ of $\mathbf{X}$, we have a given model describing the evolution of $\mathbf{Y}$ through time, defined by the HSDBN parameters that encode the conditional distribution $\mathbf{Y}_\mathbf{x}(t_{j+1})|\mathbf{Y}_\mathbf{x}(t_j)$.
In the context of metadata recovery, we are observing a sequence of realizations of $\mathbf{Y}$ which we know were all generated with the same model, *i.e.* which we know are associated to the same time series $\mathbf{Y}_\mathbf{x}$, and we wish to recover the corresponding value $\mathbf{x}$.

2.1.2 Inference approaches

Naive approach We first consider a naive approach, that enables optimal usage of efficient pre-programmed MAP inference algorithms. This naive algorithm is split in two steps

- We consider independently the $d-1$ following MAP inference sub-problems:

$$\mathbf{x}^{MAP(1)} \in \underset{\mathbf{x}}{\operatorname{argmax}} \; P(\mathbf{X} = \mathbf{x}|\mathbf{Y}(t_1) = \mathbf{y}^{(1)}, \mathbf{Y}(t_2) = \mathbf{y}^{(2)})$$
$$\ldots$$
$$\mathbf{x}^{MAP(p)} \in \underset{\mathbf{x}}{\operatorname{argmax}} \; P(\mathbf{X} = \mathbf{x}|\mathbf{Y}(t_p) = \mathbf{y}^{(p)}, \mathbf{Y}(t_{p+1}) = \mathbf{y}^{(p+1)})$$
$$\ldots$$
$$\mathbf{x}^{MAP(d-1)} \in \underset{\mathbf{x}}{\operatorname{argmax}} \; P(\mathbf{X} = \mathbf{x}|\mathbf{Y}(t_{d-1}) = \mathbf{y}^{(d-1)}, \mathbf{Y}(t_d) = \mathbf{y}^{(d)}).$$

[4]These time stamps are considered equally spaced, with the same time step Δ_t used in the dynamic part of the model $\mathcal{B}$.

This comes down to focusing on the time series observations, two successive time stamps at a time. In each of those problems the observation $(\mathbf{x}, \mathbf{y}^{(p)}, \mathbf{y}^{(p+1)})$ can be viewed as a realization of variables $(\mathbf{X}, \mathbf{Y}, \tilde{\mathbf{Y}})$, which are modeled by the HSDBN.

As a consequence, each of these problems can be rewritten as, for $p \in [\![1, d-1]\!]$,

$$\mathbf{x}^{MAP(p)} \in \underset{\mathbf{x}}{\operatorname{argmax}}\ P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p+1)}). \tag{3.4}$$

These MAP inference sub-problems can then be solved straightforwardly with a standard inference algorithm, such as `LikelihoodWeighting`, presented as Algorithm 1 in Chapter 1, using the HSDBN parameters Θ.
This generates $d-1$ MAP estimates of $\mathbf{x}$: $\{\hat{\mathbf{x}}^{MAP(1)}, \dots, \hat{\mathbf{x}}^{MAP(d-1)}\}$.

- We then choose to aggregate these results by considering the global MAP estimate.
  We make the assumption that $\hat{\mathbf{x}}^{MAP}$ has great chance of being the most represented configuration among $\{\hat{\mathbf{x}}^{MAP(1)}, \dots, \hat{\mathbf{x}}^{MAP(d-1)}\}$: we approximate $\hat{\mathbf{x}}^{MAP}$ defined in Equation (3.3) by:

$$\hat{\mathbf{x}}^{MAP(Naive)} = \underset{\mathbf{x} \in Val(\mathbf{X})}{\operatorname{argmax}} \sum_{p=1}^{d} \mathbb{I}_{\mathbf{x} = \mathbf{x}^{MAP(p)}}, \tag{3.5}$$

  where $\hat{\mathbf{x}}^{MAP(1)}, \dots, \hat{\mathbf{x}}^{MAP(d-1)}$ are the solutions of the optimization problems defined by Equation (3.4).

This approach has an important computational advantage in practice: it enables the usage of already packaged and optimized MAP inference algorithms.
However, it does not take advantage of the fact that the value of $\mathbf{x}$ is the same for all the observations $\{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(d)}\}$, and the aggregation of the MAP inference sub-problems is not mathematically sound.

Sound approach - version 1 We want to take advantage of the fact that the observations belong to a unique time series $\mathbf{Y}_{\mathbf{x}}$ associated with a single configuration of metadata $\mathbf{x} \in Val(\mathbf{X})$. For any $p \in [\![2, d-1]\!]$ and $j \in [\![1, p-1]\!]$, we know that $\mathbf{Y}(t_{p-j}) \perp \mathbf{Y}(t_{p+1}) | \{\mathbf{X}, \mathbf{Y}(t_p)\}$. This enables us to decompose the target distribution $P(\mathbf{X} | \mathbf{Y}(t_1), \dots, \mathbf{Y}(t_d))$ as follows:[5]

$$\begin{aligned} P(\mathbf{X} | \mathbf{Y}(t_1), \dots, \mathbf{Y}(t_d)) &\propto P(\mathbf{X}) P(\mathbf{Y}(t_1), \dots, \mathbf{Y}(t_d) | \mathbf{X}) \\ &= P(\mathbf{X}) \underbrace{P(\mathbf{Y}(t_d) | \mathbf{Y}(t_1), \dots, \mathbf{Y}(t_{d-1}), \mathbf{X})}_{P(\mathbf{Y}(t_d) | \mathbf{Y}(t_{d-1}), \mathbf{X})} P(\mathbf{Y}(t_1), \dots, \mathbf{Y}(t_{d-1}) | \mathbf{X}). \end{aligned}$$

[5] As explained in Section 1.2, the notation P is used as a general reference to random variables distribution. It is to be understood as either a configuration probability for categorical variables, or as a density function for continuous variables.

By immediate induction we then get:

$$P(\mathbf{X}|\mathbf{Y}(t_1),\dots,\mathbf{Y}(t_d)) \propto P(\mathbf{X}) \prod_{p=1}^{d} P(\mathbf{Y}(t_p)|\mathbf{Y}(t_{p-1}),\mathbf{X}), \tag{3.6}$$

with the convention that $\mathbf{Y}(t_0) = \emptyset$.

We can therefore decompose the problem of estimating $P(\mathbf{X}|\mathbf{Y}(t_1),\dots,\mathbf{Y}(t_d))$, into sub-problems: for every $p \in [\![1, d-1]\!]$, we want to estimate

$$P(\mathbf{Y}(t_{p+1})|\mathbf{Y}(t_p),\mathbf{X}) = \prod_{i=1}^{k} P(Y_i(t_{p+1})|\mathbf{Y}(t_p),\mathbf{X}).$$

For $p \in [\![1, d-1]\!]$, $i \in [\![1, k]\!]$ and $\mathbf{x} \in Val(\mathbf{X})$, the distribution $P(Y_i(t_{p+1})|\mathbf{Y}(t_p),\mathbf{X})$ is given by the following conditional density function, that can be rewritten using variables $\mathbf{X}$, $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ modeled by $\mathcal{B}$:

$$\begin{aligned} f_{Y_i(t_{p+1})}\left(y_i^{(p+1)}|\mathbf{Y}(t_p)=\mathbf{y}^{(p)},\mathbf{X}=\mathbf{x}\right) &= f_{\tilde{Y}_i}\left(y_i^{(p+1)}|\mathbf{Y}=\mathbf{y}^{(p)},\mathbf{X}=\mathbf{x}\right) \\ &= f_{\tilde{Y}_i}\left(y_i^{(p+1)}|\mathbf{Y}=\mathbf{y}^{(p)},\mathbf{X}=\mathbf{x}\right). \end{aligned}$$

The conditional independence relations implied by the structure G enable to rewrite the conditional densities of the $\tilde{Y}_i$s as follows:[6]

$$f_{\tilde{Y}_i}(y_i^{(p+1)}|\mathbf{Y}=\mathbf{y}^{(p)},\mathbf{X}=\mathbf{x}) = f_{\tilde{Y}_i}\left(y_i^{(p+1)}|\mathbf{Y}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})}=\mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})},\mathbf{X}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}=\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}\right).$$

These distributions can be written using the parameters Θ of $\mathcal{B}$. We remind that, for $i \in [\![1, k]\!]$, the parameters corresponding to the variable $\tilde{Y}_i$ are given by $\Theta_i^{\tilde{\mathbf{Y}}}$ where:

$$\Theta_i^{\tilde{\mathbf{Y}}} = \left\{(\beta_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}, \sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}})\right\}_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})} \in Val\left(\mathbf{X}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}\right)}.$$

Indeed, we have:

$$\begin{aligned} f_{\tilde{Y}_i}(y_i^{(p+1)}|\mathbf{Y}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})} &= \mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})}, \mathbf{X}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})} = \mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}) \\ &= \frac{1}{\sqrt{2\pi}\sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}} \exp\left(-\frac{\left(y_i^{(p+1)} - \mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})}\beta^T_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}\right)^2}{2\sigma^2_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}}\right). \end{aligned}$$

Combining all these results[7], we obtain the closed-form expression of a quantity which is proportional to our target distribution $P(\mathbf{X}|\mathbf{E})$: for $\mathbf{x} \in Val(\mathbf{X})$, and reminding that $\mathbf{E} =$

[6] This equation expresses a simpler idea than the previous one, despite being much less readable.

[7] We remind that $P(\mathbf{X}=\mathbf{x})$ can be written simply using as a discrete parameter table

$$\Theta_i^{\mathbf{X}} = \left\{\theta_{x_i|\mathbf{x}_{\pi_{\mathbf{X}}}(v_i^{\mathbf{X}})}\right\}_{x_i,\mathbf{x}_{\pi_{\mathbf{X}}}(v_i^{\mathbf{X}})}.$$

$\{\mathbf{Y}(t_1) = \mathbf{y}^{(1)}, \ldots, \mathbf{Y}(t_d) = \mathbf{y}^{(d)}\}$,

$$P(\mathbf{X} = \mathbf{x}|\mathbf{E}) \propto \prod_{i=1}^{n} \theta_{x_i|\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\mathbf{X}})}} \prod_{p=1}^{d} \prod_{i=1}^{k} \frac{1}{\sqrt{2\pi}\sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}} \exp\left(- \frac{\left(y_i^{(p+1)} - \mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})} \boldsymbol{\beta}^T_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}} \right)^2}{2\sigma^2_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}} \right). \tag{3.7}$$

In practice, we compute the logarithm of this quantity, *i.e.*

$$\begin{aligned} \log(P(\mathbf{X} = \mathbf{x}|\mathbf{E})) = K &+ \sum_{i=1}^{n} \log\left(\theta_{x_i|\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\mathbf{X}})}}\right) \\ &- \sum_{p=1}^{d} \sum_{i=1}^{k} \left[\log\left(\sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}\right) + \left(\frac{\left(y_i^{(p+1)} - \mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})} \boldsymbol{\beta}^T_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}} \right)^2}{2\sigma^2_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}} \right) \right]. \end{aligned} \tag{3.8}$$

where K is a constant that is independent of $\mathbf{x}$.

We consider that computing $P(\mathbf{X} = \mathbf{x}|\mathbf{E})$ for each $\mathbf{x}$ in $Val(\mathbf{X})$ does not raise any computational problem, since the number of accessible values of $\mathbf{X} = (X_1, \ldots, X_n)$ is assumed to be reasonable in practice.

Sound approach - version 2 The previously proposed approach requires to quickly access the HSDBN parameters to compute expression in Equation (3.7). Some efficiency issues were faced programming this inference. We therefore explored another possibility: a simple rewriting trick using Bayes formula enables to link the distribution $P(\mathbf{X}|\mathbf{E})$ decomposed in Equation (3.6) into several inference sub-problems also targeting the conditional distribution of the metadata variables $\mathbf{X}$, given evidence from successive realizations of $\mathbf{Y}$ only. Indeed:

$$\begin{aligned} P(\mathbf{X}|\mathbf{Y}(t_1), \ldots, \mathbf{Y}(t_d)) &\propto P(\mathbf{X}) \prod_{p=1}^{d} P(\mathbf{Y}(t_p)|\mathbf{Y}(t_{p-1}), \mathbf{X}) \\ &= P(\mathbf{X}) \prod_{p=1}^{d} \frac{P(\mathbf{Y}(t_p), \mathbf{Y}(t_{p-1})|\mathbf{X})}{P(\mathbf{Y}(t_{p-1})|\mathbf{X})} \\ &= P(\mathbf{X}) \prod_{p=1}^{d} \frac{P(\mathbf{X}|\mathbf{Y}(t_p), \mathbf{Y}(t_{p-1}))}{P(\mathbf{X}|\mathbf{Y}(t_{p-1}))} \underbrace{\frac{P(\mathbf{Y}(t_p)|\mathbf{Y}(t_{p-1}))}{}}_{\text{indep of } \mathbf{X}} \\ &\propto P(\mathbf{X}) \prod_{p=1}^{d} \frac{P(\mathbf{X}|\mathbf{Y}(t_p), \mathbf{Y}(t_{p-1}))}{P(\mathbf{X}|\mathbf{Y}(t_{p-1}))} \end{aligned}$$

In this second version of the sound approach, we need to estimate, for $p \in [\![1, d]\!]$,

$$\frac{P(\mathbf{X}|\mathbf{Y}(t_p), \mathbf{Y}(t_{p-1}))}{P(\mathbf{X}|\mathbf{Y}(t_{p-1}))}. \tag{3.9}$$

This can be rewritten using the variables modeled by the HSDBN $\mathcal{B} = (G, \Theta)$. For $p \in [\![1, d]\!]$ and $\mathbf{x} \in Val(\mathbf{X})$,

$$\frac{P(\mathbf{X} = \mathbf{x} | \mathbf{Y}(t_p) = \mathbf{y}^{(p)}, \mathbf{Y}(t_{p-1}) = \mathbf{y}^{(p-1)})}{P(\mathbf{X} = \mathbf{x} | \mathbf{Y}(t_{p-1}) = \mathbf{y}^{(p-1)})} = \frac{P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p)})}{P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)})}.$$

Like in the first version of the sound approach to metadata recovery, we have derived a quantity which is proportional to our target distribution $P(\mathbf{X}|\mathbf{E})$, and which only involves the variables $\mathbf{X}$, $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ modeled by $\mathcal{B}$: for $\mathbf{x} \in Val(\mathbf{X})$, and reminding that $\mathbf{E} = \{\mathbf{Y}(t_1) = \mathbf{y}^{(1)}, \ldots, \mathbf{Y}(t_d) = \mathbf{y}^{(d)}\}$,

$$P(\mathbf{X} = \mathbf{x} | \mathbf{E}) \propto P(\mathbf{X} = \mathbf{x}) \prod_{p=1}^{d} \frac{P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p)})}{P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)})}. \tag{3.10}$$

Like in the first approach, we compute the log of this quantity in practice, *i.e.*

$$\begin{aligned} \log(P(\mathbf{X} = \mathbf{x} | \mathbf{E})) &= K' + \log(P(\mathbf{X} = \mathbf{x})) \\ &\quad + \sum_{p=1}^{d} \left[\log\left(P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p)})) - \log(P(\mathbf{X} = \mathbf{x} | \mathbf{Y} = \mathbf{y}^{(p-1)}) \right) \right], \end{aligned} \tag{3.11}$$

where K' is a constant independent of $\mathbf{x}$.

Using these results, the target distribution $P(\mathbf{X}|\mathbf{E})$ can be evaluated by estimating p times the distributions $P(\mathbf{X}|\mathbf{Y}, \tilde{\mathbf{Y}})$ and $P(\mathbf{X}|\mathbf{Y})$, which can be done using the Bayesian network $\mathcal{B}$ and a standard inference algorithm such as `LikelihoodWeighting`.

2.1.3 Inference algorithms presentation

Using the derivations made in the last subsection, we will now present the following inference algorithms:

1. The `NaiveMetadataRecovery` algorithm, which takes on the idea of the naive approach to perform MAP inference, relying on Equation (3.5).
2. The `SoundMetadataRecovery.v1` algorithm, which uses Equation (3.8) to perform inference through direct computations using the parameters of the HSDBN.
3. The `SoundMetadataRecovery.v2` algorithm, which uses Equation (3.11) to perform inference by aggregation of the results of inference sub-problems using observations at successive time stamps.

The inputs of these three inference algorithms are among:

- $\mathcal{B} = (G, \Theta)$ a hybrid static-dynamic BN modeling $\mathbf{X} = (X_1, \ldots, X_n)$ (static variables) and $\mathbf{Y} = (Y_1, \ldots, Y_k)$ (dynamic variables observed on $\{t_1, \ldots, t_p\}$),

- $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)}$: a sequence of successive realizations of $\mathbf{Y}$, that correspond to a given value $\mathbf{x}$ of $\mathbf{X}$,
- `sota-MAPInference`: a standard MAP inference algorithm, taking as input `BN`: a Bayesian network, `evidence`: information concerning part of the variables modeled by the BN, and `target`: a tuple of categorical variables, and returning the estimated MAP configuration for `target` given `evidence`.
- `sota-Inference`: a standard inference algorithm, taking as input `BN`: a Bayesian network, `evidence`: information concerning part of the variables modeled by the BN, and `target`: a tuple of categorical variables, and returning a set of estimated probabilities for all configurations of `target` given `evidence`.

Algorithm 9 `NaiveMetadataRecovery`

Input: $\mathcal{B} = (G, \Theta)$, $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)}$, `sota-MAPInference`

for $p = 1$ to $d - 1$ **do**
  $\mathbf{E}_p \leftarrow \{\mathbf{Y} = \mathbf{y}^{(p)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p+1)}\}$
  $\hat{\mathbf{x}}^{MAP(p)} \leftarrow$ `sota-MAPInference`$(\texttt{BN} = \mathcal{B}, \texttt{evidence} = \mathbf{E}_p, \texttt{target} = \mathbf{X})$
$\hat{\mathbf{x}}^{MAP(Naive)} \leftarrow \underset{\mathbf{x}}{\operatorname{argmax}} \sum_{p=1}^{d} \mathbb{I}_{\{\mathbf{x}=\hat{\mathbf{x}}^{MAP(p)}\}}$

Output: $\hat{\mathbf{x}}^{MAP(Naive)}$

Algorithm 10 `SoundMetadataRecovery.v1`

Input: $\mathcal{B} = (G, \Theta)$, $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)}$

$\{\texttt{logP}_{\mathbf{x}}\} \leftarrow \{0\}_{\mathbf{x}}$
for $\mathbf{x} \in Val(\mathbf{X})$ **do** *#for configurations* $\mathbf{x}$ *of* $\mathbf{X}$*, compute* $P(\mathbf{X} = \mathbf{x}|\mathbf{E})$
  for $i = 1$ to n **do**
    $\texttt{logP}_{\mathbf{x}} \leftarrow \texttt{logP}_{\mathbf{x}} + \log(\theta_{x_i|\mathbf{x}_{\pi_{\mathbf{X}}}(v_i^{\mathbf{x}})})$
  for $p = 1$ to $d - 1$ **do**
    for $i = 1$ to k **do**
      $\texttt{logP}_{\mathbf{x}} \leftarrow \texttt{logP}_{\mathbf{x}} + \log\left(\sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}\right) + \left(\frac{(y_i^{(p+1)} - \mathbf{y}^{(p)}_{\pi_{\mathbf{Y}}(v_i^{\tilde{\mathbf{Y}}})} \boldsymbol{\beta}^T_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}})^2}{2\sigma_{\mathbf{x}_{\pi_{\mathbf{X}}(v_i^{\tilde{\mathbf{Y}}})}}}\right)$
$\{w_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\exp(\texttt{logP}_{\mathbf{x}})\}_{\mathbf{x}}$
$Z \leftarrow \sum_{\mathbf{x}} w_{\mathbf{x}}$
$\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\frac{w_{\mathbf{x}}}{Z}\}_{\mathbf{x}}$ *#normalize the* $w_{\mathbf{x}}$ *so they sum to* 1

Output: $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}}$

Algorithm 11 `SoundMetadataRecovery.v2`

Input: $\mathcal{B} = (G, \Theta)$, $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(d)}$, `sota-Inference`

1: $\{\texttt{logP}_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{0\}_{\mathbf{x}}$
2: **for** $i = 1$ to n **do**
3: $\quad \{\texttt{logP}_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\texttt{logP}_{\mathbf{x}} + \log(\theta_{x_i|\mathbf{x}_{\pi_{\mathbf{X}}}(v_i^{\mathbf{x}})})\}_{\mathbf{x}}$
4: **for** $p = 1$ to $d-1$ **do** *#for all time stamps t_p, compute $P(\mathbf{X}|\mathbf{Y}(t_p), \mathbf{Y}(t_{p-1}))$ and $P(\mathbf{X}|\mathbf{Y}(t_{p-1}))$*
5: $\quad \mathbf{E}_p^{(1)} \leftarrow \{\mathbf{Y} = \mathbf{y}^{(p)}, \tilde{\mathbf{Y}} = \mathbf{y}^{(p+1)}\}$
6: $\quad \mathbf{E}_p^{(2)} \leftarrow \{\mathbf{Y} = \mathbf{y}^{p}\}$
7:
8: $\quad \{p_{\mathbf{x}}^{(1)}\}_{\mathbf{x}} \leftarrow \texttt{sota-Inference}(\texttt{BN} = \mathcal{B}, \texttt{evidence} = \mathbf{E}_p^{(1)}, \texttt{target} = \mathbf{X})$
9: $\quad \{p_{\mathbf{x}}^{(2)}\}_{\mathbf{x}} \leftarrow \texttt{sota-Inference}(\texttt{BN} = \mathcal{B}, \texttt{evidence} = \mathbf{E}_p^{(2)}, \texttt{target} = \mathbf{X})$
10: $\quad \{\texttt{logP}_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\texttt{logP}_{\mathbf{x}} + \log(p_{\mathbf{x}}^{(1)}) - \log(p_{\mathbf{x}}^{(2)})\}_{\mathbf{x}}$
11: $\{w_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\exp(\texttt{logP}_{\mathbf{x}})\}_{\mathbf{x}}$
12: $Z \leftarrow \sum_{\mathbf{x}} w_{\mathbf{x}}$
13: $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}} \leftarrow \{\frac{w_{\mathbf{x}}}{Z}\}_{\mathbf{x}}$ *#normalize the $w_{\mathbf{x}}$ so they sum to* 1

Output: $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}}$

The subscript $\mathbf{x}$, on lines $1, 8, 9$ and 10 of Algorithm 10, and lines $1, 3$ and to 13 of Algorithm 11, denotes a vector of objects that are indexed by $\mathbf{x}$, for all $\mathbf{x} \in Val(\mathbf{X})$).

In practice, `SoundMetadataRecovery.v2` is more efficient than `SoundMetadataRecovery.v1`. We have identified a few insights on why this is observed:

- First, Algorithm 11 implies only manipulations of vectors of length $|Val(\mathbf{X})|$, which is efficient in practice: the operations on lines $8-10$ can be vectorized.[8] In Algorithm 10 however, the for loop on line 6 cannot be transformed in a vector operation efficiently.
- Moreover, typical particle-based `sota-Inference` algorithms, such as `LikelihoodWeighting`, despite being described as a way to evaluate the probability of a given configuration $\mathbf{x}_Q$ of query variables $\mathbf{X}_Q$ in Chapter 1, are very well suited to the simultaneous evaluation of the entire distribution $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}}$ as depicted, as it is done on lines $8-9$ of Algorithm 11. This enables to estimate $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}}$ without explicitly having a for loop going through every value of $\mathbf{X}$ as it is the case in Algorithm 10.
- Lastly, `LikelihoodWeighting` is already efficiently programmed and packaged, whereas the computation of the closed form expression given by Equation (3.8) in Algorithm 10 is slower than expected.

[8] we remind that the number of accessible values of $\mathbf{X}$ is equal to the number of rows of a typical metadata table $D^{\mathbf{X}}$ (which is far smaller than $\prod_{i=1}^{n} |Val(X_i)|$ in practice since the redundancy and determinism imply that the majority of the configurations $\mathbf{x} \in Val(X_1) \times \cdots \times Val(X_n)$ are not accessible).

2.1.4 Generalization: partial recovery of metadata, partially observed metadata

In practice, there may be two main alterations to the setting of metadata recovery we proposed in the previous subsections.

1. We may know part of the metadata associated with the observed time series, *i.e.* there exist $E \subset [\![1, n]\!]$ and $\mathbf{x}_E^{ev} \in Val(\mathbf{X}_E)$ such that $\{\mathbf{X}_E = \mathbf{x}_E^{ev}\}$ belongs to the evidence.

2. We may want to recover only part of the metadata, *i.e.* we are only interested in the distribution of $\mathbf{X}_Q$, where $Q \subset [\![1, n]\!]$, given evidence.

The general problem of metadata recovery comes down to the estimation of the distribution:

$$P(\mathbf{X}_Q | \mathbf{E}, \mathbf{X}_E = \mathbf{x}_E^{ev}) \tag{3.12}$$

where $\mathbf{E} = \{\mathbf{Y}(t_1) = \mathbf{y}^{(1)}, \dots, \mathbf{Y}(t_d) = \mathbf{y}^{(d)}\}$, as presented in Equation (3.2).

The problem of estimating the distribution given by Equation (3.12) can be narrowed down to the problem we tackled in the previous subsection.
Let $E, Q \subset [\![1, n]\!]$, $J = [\![1, n]\!] \setminus (Q \cup E)$ and $\mathbf{x}_E^{ev} \in Val(\mathbf{X}_E)$. We can write, $\forall \mathbf{x}_Q \in Val(\mathbf{X}_Q)$:

$$\begin{aligned}
P(\mathbf{X}_Q = \mathbf{x}_Q | \mathbf{E}, \mathbf{X}_E = \mathbf{x}_E^{ev}) &= \frac{P(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E = \mathbf{x}_E^{ev} | \mathbf{E})}{P(\mathbf{X}_E = \mathbf{x}_E^{ev}) | \mathbf{E})} \\
&\propto P(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E = \mathbf{x}_E^{ev} | \mathbf{E}) \\
&= \sum_{\mathbf{x}_J \in Val(\mathbf{X}_J)} P(\mathbf{X}_J = \mathbf{x}_J, \mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E = \mathbf{x}_E^{ev} | \mathbf{E}) \\
&= \sum_{\mathbf{x}_J \in Val(\mathbf{X}_J)} P(\underbrace{\mathbf{X}_{J \cup Q \cup E}}_{\text{reorg. of } \mathbf{X}} = (\mathbf{x}_J, \mathbf{x}_Q, \mathbf{x}_E^{ev}) | \mathbf{E}).
\end{aligned}$$

Computing each of the terms of the last sum boils down to the estimation of the probability of observing the value $\mathbf{x}$ of $\mathbf{X}$, given evidence on temporal variables only.
In practice, partial query with evidence on $\mathbf{X}$ comes down to filtering and summing rows of $\{\hat{p}_{\mathbf{x}}\}_{\mathbf{x}}$, output of Algorithm 10 and 11.

Now that we are able to use a HSDBN model $\mathcal{B}$ to perform inference, we are interested in learning it from data. In the next subsection, we propose a learning algorithm, using `ds-BNSL` algorithm and a state of the art structure learning algorithm `sota-BNSL`.

2.2 Learning

2.2.1 Extra constraints for standard structure learning algorithms

By default, a baseline algorithm such as the one we refer to as `sota-BNSL` in Section 3 and 4 of Chapter 2 takes as input a dataset, and returns a Bayesian network structure. However, there

exist optional inputs that can be simply integrated into the algorithm, and that are used to define constraints on the final structure.
For the definition of our proposed HSDBN structure learning algorithm, we need to specify two of such optional inputs:[9]

- `whitelist`: list of arcs, which are **required** to be in the final structure.
- `blacklist`: list of arcs, which are **forbidden** in the final structure. For simplification purposes, this argument is given by a list of tuples (I, J) where I and J are sets of nodes: $(I, J) \in$ `blacklist` implies that all arcs (i, j) where $i \in I$ and $j \in J$ are forbidden.

The inclusion of both these inputs into a greedy local-search Bayesian network structure learning algorithm such as `HillClimbing` is straightforward: one just has to initiate the heuristic with the structure containing only the arcs present in `whitelist`, and perform standard greedy hill climbing with a slightly reduced set of possible local operators: arcs from `whitelist` cannot be removed, and arcs from `blacklist` cannot be added.

2.2.2 HSDBN structure Learning algorithm

We propose an HSDBN structure learning algorithm which follows 2 steps:

1. the structure of the Bayesian network modeling only the metadata variables $\mathbf{X}$ is learned by running `ds-BNSL`, defined in Algorithm 6 in Chapter 2, with $D^{\mathbf{X}}$ as an input,
2. a standard Bayesian network structure learning algorithm `sota-BNSL`, that handles mixed continuous and categorical data, is run on $D^{\mathbf{X}\mathbf{Y}\tilde{\mathbf{Y}}}$ with the following two constraints:
   - the constraints imposed by the definition of a HSDBN in Section 1.2 must be verified,
   - no supplementary arc can be learned in between nodes $\{v_i^{\mathbf{X}}\}$, *i.e.* the only arcs linking two nodes representing metadata variables are the ones that have been learned by `ds-BNSL` during the algorithm's first step.

These two constraints can be encoded into the `whitelist` and `blacklist` arguments. If $G^{\mathbf{X}} = (V^{\mathbf{X}}, A^{\mathbf{X}})$ is the graph that was learned during the algorithm's first step, we set:

- `whitelist` $= A^{\mathbf{X}}$,
- `blacklist` $= \{(V^{\mathbf{X}}, V^{\mathbf{X}}), (V^{\tilde{\mathbf{Y}}}, V^{\mathbf{Y}}), (V^{\mathbf{Y}}, V^{\mathbf{X}}), (V^{\tilde{\mathbf{Y}}}, V^{\mathbf{X}}), (V^{\mathbf{Y}}, V^{\mathbf{Y}}), (V^{\tilde{\mathbf{Y}}}, V^{\tilde{\mathbf{Y}}})\}$.

We now present the hybrid static-dynamic BN structure learning algorithm described in the previous subsection. Using the notations introduced in Section 1, this algorithm takes as input

- $D^{\mathbf{X}}$: a dataset containing M observations of $\mathbf{X}$,

[9] The name of these arguments is borrowed from the `bnlearn` package.

- $D^{\mathbf{XY\tilde{Y}}}$: a dataset containing observations of $\mathbf{X}$, as well as the observations of $\mathbf{Y}$ on two successive time stamps,
- `sota-BNSL`: a standard static Bayesian network structure learning algorithm (typically close to state of the art), taking for input `data`: a dataset containing observations of the variables which we want to model, optional `whitelist` and `blacklist` arguments, and returning a BN structure.

Algorithm 12 `HybridStaticDynamicBNSL`

Input: $D^{\mathbf{X}}$, $D^{\mathbf{XY\tilde{Y}}}$, `sota-BNSL`
1: $G^{\mathbf{X}} = (V^{\mathbf{X}}, A^{\mathbf{X}}) \leftarrow$ `ds-BNSL`$(D^{\mathbf{X}}$, `sota-BNSL`$)$
`bl` $\leftarrow \{(V^{\mathbf{X}}, V^{\mathbf{X}}), (V^{\tilde{\mathbf{Y}}}, V^{\mathbf{Y}}), (V^{\mathbf{Y}}, V^{\mathbf{X}}), (V^{\tilde{\mathbf{Y}}}, V^{\mathbf{X}}), (V^{\mathbf{Y}}, V^{\mathbf{Y}}), (V^{\tilde{\mathbf{Y}}}, V^{\tilde{\mathbf{Y}}})\}$
2: $G \leftarrow$ `sota-BNSL`$(D^{\mathbf{XY\tilde{Y}}}$, `whitelist` $= A^{\mathbf{X}}$, `blacklist` $=$ `bl`$\})$
Output: G

2.2.3 Parameter learning

The hybrid static-dynamic Bayesian network model we proposed is a Bayesian network modeling variables $\mathbf{X}, \mathbf{Y}$ and $\tilde{\mathbf{Y}}$, observed in the dataset $D^{\mathbf{XY\tilde{Y}}}$.
Since we are in the case of complete data, parameter learning may be done in a straightforward way using maximum likelihood estimation, as presented in Chapter 1, for networks modeling both categorical and continuous variables.

3 Hybrid static-dynamic Bayesian networks in practice: not everything can be learned

In this section, we describe the data that are available in practice from an IoT system, and we explain what choices have to be made in order to obtain the setting described in Section 1.1, which is essential for being able to learn a hybrid static-dynamic Bayesian network.

3.1 From real data to our formal setting

3.1.1 Description of observed datasets

We consider that the data coming from an IoT system can be represented as two tables, closely related to the datasets $D^{\mathbf{X}}$ and $D^{\mathbf{XY}}$ presented in Section 1.1.2.

Metadata table The metadata table corresponding to an IoT system is typically backed by relational or semantic databases, and contains observations of n categorical variables including an **id** variable, which is the **time series unique identifier** variable. Such a variable has a unique value for every row of the table and entirely defines a given time series, for this reason we

also call it **key**.
Using the previously introduced notations this table is defined by a dataset $D^{\mathbf{X}}$, containing M distinct observations of a tuple of categorical variables $\mathbf{X} = (X_1, \ldots, X_n)$. For simplicity, we consider that X_1 is the time series identification variable, *i.e.* it satisfies, for $1 \leq m, m' \leq M$,

$$x_1^{(m)} = x_1^{(m')} \Rightarrow m = m'.$$

$D^{\mathbf{X}}$ therefore obviously satisfies the metadataset assumption introduced in Section 1.1.2.

Time series observations table The time series observations table contains values of time series measured in the associated system. This table contains three columns corresponding to: the time stamp, the time series identifier, and the measured value of the corresponding (univariate) time series at that given time stamp.
Consistently, with the previously defined notations, this table is denoted by $D^{X_1 Y}$.

3.1.2 Synchronization and choice of time step

Time series synchronization: bin size A priori, all temporal variables are not observed simultaneously: the sequence of time stamps $\{t_1^{x_1}, \ldots, t_{l_{x_1}}^{x_1}\}$ and its length depend on the considered variable Y_{x_1}, identified by the value x_1 of the key metadata variable X_1. We may however synchronize the associated time series observations by choosing a **binsize** value δ_t, a starting time stamp t_1, and a number of bins L such that every temporal variable is observed at least once in each time window $]t_1 + (p-1)\delta_t, t_1 + p\delta_t]$ for $p \in [\![1, L]\!]$. The observation of every given temporal variable in each time window are then averaged, resulting in M time series measured on the same sets of time stamps:

$$\{\{Y_{x_1}(t_1 + p\delta_t)\}_{0 \leq p \leq L}\}_{x_1 \in Val(X_1)}.$$

Note that many other aggregation methods exist and are equally applicable, in all that follows, we will consider that we always have synchronized time series observations.

Time series Markovian modeling: time step Assuming all temporal variables observations in $D^{X_1 Y}$ are synchronized, we may now choose a value for the **time step** Δ_T ($\geq \delta_t$), in order to define the sequence of time stamps $\{t_p = t_1 + (p-1)\Delta_t\}_{1 \leq p \leq l}$ that are used for modeling purposes.
The measured values of every given temporal variables in each time window $[t_p, t_{p+1}]$ is then averaged, in order to obtain synchronized time series observations on the sequence of time stamps $t_1, \ldots t_l$, in accordance with the setting presented in Section 1.
In practice, this value is chosen according to the typical time of variation of the measured

quantities. We should also make sure that the selected value of Δ_t is consistent with the Markovian assumption (assumption A-1).

3.1.3 Balancing static and temporal data

From univariate to multivariate time series In real-world metadatasets, considering all the available metadata variables implies that the time series corresponding to a given metadata configuration is univariate, *i.e.* $k = 1$ in Equation (3.1). However, we are often interested in modeling multivariate time series that represent the dependencies in the evolution of quantities that we believe to be linked (as it is the case in example ($\star$), for the temperature and the CO_2 level in a given location).
For this purpose, we need to make the choice of an observation **scope**, corresponding to the temporal variables we wish to observe simultaneously as a multivariate variable $\mathbf{Y} = (Y_1, \ldots, Y_k)$ introduced in Section 1.1.1.

Orthogonal dimensions in metadata In practice, this can only be done if the metadata table $D^{\mathbf{X}}$ is a Cartesian product of datasets containing the values of disjoint subsets of variables, that we call **orthogonal dimensions**.
Formally, for a tuple of categorical variables $\mathbf{X} = (X_1, \ldots, X_n)$ observed in a dataset $D^{\mathbf{X}}$ satisfying the metadataset assumption[10] introduced in Section 1.1.2, this supposes the existence of a partition $I_1, \ldots, I_q$ of $[\![1, n]\!]$, with $q \geq 2$[11], such that:

$$Val(\mathbf{X}) = \bigtimes_{j=1}^{q} Val\left(\mathbf{X}_{I_j}\right). \tag{3.13}$$

The $\mathbf{X}_{I_j}$s for $j \in [\![1, q]\!]$ are the **orthogonal dimensions** of the metadata.
For instance, a tuple $\mathbf{X}_{I_j}$ for a given $j \in [\![1, q]\!]$ may contain all the variables describing:

- the **type** of the measured quantity (temperature, number of people, ...),
- the **equipment** the measured quantity relates to (heat pump, air handling unit, ...),
- the **location** of the measured quantity (office, openspace, ...),
- the **measuring device**.

[10] This implies that $D^{\mathbf{X}}$ contains exactly $Val(\mathbf{X})$.
[11] This notably implies that the dataset does not contain an identifier variable that has a different value in every row. As we will see in the example, such a variable is generally discarded to perform the modeling.

Choosing the scope: removing a dimension Assuming that $I_1, \dots, I_q$ satisfying equation 3.13 exist, the choice of the observation **scope** narrows down to the choice a dimension to **remove** from the metadataset $\mathbf{X}$. This dimension describes the temporal variables that are observed simultaneously as a multivariate temporal variable $\mathbf{Y}$.
Formally, we suppose $\bar{I} = I_q$ is the set of indexes corresponding to the variables we wish to remove from the metadata, and $I = \bigcup_{j=1}^{q-1} I_j$ is the set of indexes corresponding to the remaining metadata variables. The different configurations of $\mathbf{X}_{\bar{I}}$ describe the simultaneously observed temporal variables: instead of a single temporal variable Y observed for each configuration of the metadata $\mathbf{X}$, there are now $k = |Val(\mathbf{X}_{\bar{I}})|$ temporal variables observed in parallel for each configuration of the remaning metadata variables $\mathbf{X}_I$.
Our new setting is therefore made of:

- a new tuple of metadata variables: $\mathbf{X}_I$, with $|Val(\mathbf{X}_{\bar{I}})| = \frac{Val(\mathbf{X})}{k}$
- the corresponding collection of (multivariate) temporal variables $\{\mathbf{Y}_{\mathbf{x}_I}\}_{\mathbf{x}_I \in Val(\mathbf{X}_I)}$. Denoting by $\{\mathbf{x}_{\bar{I}}^1, \dots, \mathbf{x}_{\bar{I}}^k\}$ the k distinct elements of $Val(\mathbf{X}_{\bar{I}})$, then for all $\mathbf{x}_I \in Val(\mathbf{X}_I)$,

$$\mathbf{Y}_{\mathbf{x}_I} = (Y_{(\mathbf{x}_{\bar{I}}^{(1)}, \mathbf{x}_I)}, \dots, Y_{(\mathbf{x}_{\bar{I}}^{(k)}, \mathbf{x}_I)}),$$

  which correspond to the variables $Y_{\mathbf{x},1}, \dots, Y_{\mathbf{x},k}$ introduced in Equation (3.1).

In the next subsection, we use the example ($\star$) to explain this idea more clearly.

3.2 Example: from the data available in practice to the HSDBN setting

3.2.1 Available data in the case of example ($\star$)

We present an example of a metadata table and an associated time series observations table that are available from a virtual IoT system corresponding to example ($\star$) presented in Section 1.1.3.

We still consider a small part of a building, containing six zones (given by variable X_2) spread upon 3 rooms (given by variable X_3), in each of which we observe quantities with three different data types (given by variable X_4).
Table 3.5 represents the associated metadata table $D^{\mathbf{X}}$.

Discarding the identifier variable Like we explained in the previous subsection, we discard the `DataKey` variable X_1 from the metadataset, as it does not present any statistical interest. Denoting by $\mathbf{X}' = \mathbf{X}_{\{2,3,4\}}$, we notice that the dataset $D^{\mathbf{X}'}$ satisfies the metadataset assumption, that is, the rows of the dataset still constitute a valid time series identifier. Indeed, the `Zone` variable X_2 and `DataType` variable X_4 uniquely define the rows of $D^{\mathbf{X}}$.

X_1 (DataKey)	X_2 (Zone)	X_3 (Room)	X_4 (DataType)
1	Perch1	OpenSpace	Temperature
2	Perch2	OpenSpace	Temperature
3	Perch3	OpenSpace	Temperature
4	MeetRoomEast	MeetingRoom	Temperature
5	MeetRoomWest	MeetingRoom	Temperature
6	Box	Box	Temperature
7	Perch1	OpenSpace	CO_2 level
8	Perch2	OpenSpace	CO_2 level
9	Perch3	OpenSpace	CO_2 level
10	MeetRoomEast	MeetingRoom	CO_2 level
11	MeetRoomWest	MeetingRoom	CO_2 level
12	Box	Box	CO_2 level
13	Perch1	OpenSpace	NumberPeople
14	Perch2	OpenSpace	NumberPeople
15	Perch3	OpenSpace	NumberPeople
16	MeetRoomEast	MeetingRoom	NumberPeople
17	MeetRoomWest	MeetingRoom	NumberPeople
18	Box	Box	NumberPeople

Table 3.5: Metadata table $D^{\mathbf{X}}$ obtained from a virtual IoT system consistent with example ($\star$).

Identifying orthogonal dimensions in metadata Let $I_1 = \{2, 3\}$ and $I_2 = \{4\}$. Considering the values observed in the dataset $D^{\mathbf{X}'}$ (corresponding to the three last columns of Table 3.5, also displayed in Table 3.7) only, we have:

$$Val(\mathbf{X}_{\{2,3,4\}}) = Val\left(\mathbf{X}_{I_1}\right) \times Val\left(\mathbf{X}_{I_2}\right). \tag{3.14}$$

This shows that $\mathbf{X}_{I_1}$ (`Zone` and `Room`) and $\mathbf{X}_{I_2}$ (`DataType`) represent two **orthogonal dimensions** as introduced in Equation (3.13): location and data type respectively.

We now present an example of a time series observations table D^{X_1Y}, displayed in Table 3.6. For simplicity, we assume we already did the synchronization (binning) as well as the time step choice, such that all time series are now synchronized on time stamps $t_1, \ldots, t_l$, equally spaced by a given value Δ_t (typically 15min).

As explained in Section 3.1.3, we can now choose the **scope** of our model, enabling the simultaneous modeling of several temporal variables as a multivariate time series. In the next subsections, we propose two different scopes, and the associated datasets $D^{\mathbf{X}}$ and $D^{\mathbf{XY}}$ corresponding to the HSDBN setting presented in Section 1.1.

3.2.2 Balancing static and temporal data: single time series scope

The most simple scope we can choose is the single time series scope:

	X_1 (DataKey)	Y (Value)
t_1	1	2.0
t_2	1	3.0
...	...	...
t_{l-1}	1	1.0
t_l	1	2.0
...	...	...
t_1	5	0.0
t_2	5	8.0
...	...	...
t_{l-1}	5	10.0
t_l	5	11.0
...	...	...
t_1	18	0.0
t_2	18	8.0
...	...	...
t_{l-1}	18	10.0
t_l	18	11.0

Table 3.6: Extract of the time series observations table $D^{X_1 Y}$ corresponding to a virtual IoT system consistent with example ($\star$)

- The metadata variables in $\mathbf{X}$ uniquely define a single time series. This corresponds to considering the table $D^{\mathbf{X}}$ presented in Table 3.7.
- For $\mathbf{x} \in Val(\mathbf{X})$, $Y_{\mathbf{x}}$ is a single temporal variable, associated with a univariate time series $\{Y_{\mathbf{x}}(t_j)\}_{1 \leq j \leq l}$. The corresponding table $D^{\mathbf{X}Y}$ is displayed in Table 3.8.

Figure 3.3 displays an example structure of a HSDBN learned on dataset $D^{\mathbf{X}Y}$.

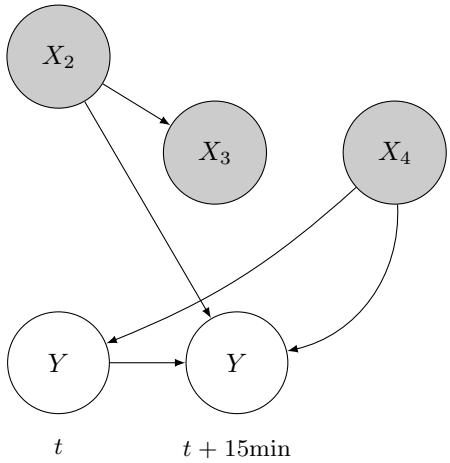


Figure 3.3: Example structure of a hybrid static dynamic Bayesian network corresponding to example ($\star$) considered at the time series scope. Static nodes are colored in light gray and temporal nodes in white.

X_2 (Zone)	X_3 (Room)	X_4 (DataType)
Perch1	OpenSpace	Temperature
Perch2	OpenSpace	Temperature
Perch3	OpenSpace	Temperature
MeetRoomEast	MeetingRoom	Temperature
MeetRoomWest	MeetingRoom	Temperature
Box	Box	Temperature
Perch1	OpenSpace	CO_2 level
Perch2	OpenSpace	CO_2 level
Perch3	OpenSpace	CO_2 level
MeetRoomEast	MeetingRoom	CO_2 level
MeetRoomWest	MeetingRoom	CO_2 level
Box	Box	CO_2 level
Perch1	OpenSpace	NumberPeople
Perch2	OpenSpace	NumberPeople
Perch3	OpenSpace	NumberPeople
MeetRoomEast	MeetingRoom	NumberPeople
MeetRoomWest	MeetingRoom	NumberPeople
Box	Box	NumberPeople

Table 3.7: Metadataset $D^{\mathbf{X}}$ corresponding to the single time series scope in the case of example ($\star$)

3.2.3 Balancing static and temporal data: data type scope

Choosing the `DataType` scope corresponds to the the example presented in Section 1.1.3. In this case, we choose to observe simultaneously temporal variables corresponding to all configurations of variable `DataType` (CO_2 level, temperature and number of people) and a fixed value of the remaining variables `Zone` and `Room`. For this purpose, we remove the variable X_4 (`DataType`) from the metadata variables.

We have this option in the case of example ($\star$) thanks to the fact that Equation (3.14) holds, guaranteeing that every data type (CO_2 level, temperature and number of people) is observed in every zone.

The datasets $D^{\mathbf{X}}$ and $D^{\mathbf{XY}}$ corresponding to this choice of scope are presented in Table 3.1 and Table 3.2 in Section 3.1.3. An example of associated HSDBN structure is displayed in Figure 3.2 in Section 1.1.2.

Another possible choice of scope Choosing the `Zone` scope implies that we observe simultaneously temporal variables corresponding to all the configurations of variable `Zone` and to a fixed value of variable `DataType`. We would therefore observe simultaneously values corresponding to a given data type (CO_2 level, temperature and number of people) across all zones. This seems to

	X_2 (Zone)	X_3 (Room)	X_4 (DataType)	Y (Value)
t_1	Perch1	OpenSpace	Temperature	2.0
t_2	Perch1	OpenSpace	Temperature	3.0
...	...	...	...	...
t_{l-1}	Perch1	OpenSpace	Temperature	1.0
t_l	Perch1	OpenSpace	Temperature	2.0
...	...	...	...	...
t_1	MeetRoomWest	MeetingRoom	Temperature	0.0
t_2	MeetRoomWest	MeetingRoom	Temperature	8.0
...	...	...	...	...
t_{l-1}	MeetRoomWest	MeetingRoom	Temperature	10.0
t_l	MeetRoomWest	MeetingRoom	Temperature	11.0
...	...	...	...	...
t_1	Box	Box	NumberPeople	0.0
t_2	Box	Box	NumberPeople	8.0
...	...	...	...	...
t_{l-1}	Box	Box	NumberPeople	10.0
t_l	Box	Box	NumberPeople	11.0

Table 3.8: Extract of the time series dataset $D^{\mathbf{X}Y}$ corresponding to the single time series scope in the case of example ($\star$)

make less sense physically, as it is more difficult to imagine how physical quantities in different areas can influence each other.

3.3 Including time information

3.3.1 Time information variables

The assumptions in terms of conditional independence inherent to the proposed structure of a HSDBN are reasonable in practice. However, some other variables than the metadata variables $\mathbf{X}$ and the immediate past of variables $\mathbf{Y}$ have an important impact on $\mathbf{Y}$, and need only minimal prior knowledge to be included in the model: these are the **time information** variables, which describe the current time stamp $t \in \{t_1, \dots t_l\}$. It is also often knwon as the time dimension.

Examples of such variables are:

- `DayOfWeek`: categorical variable with 7 configurations: {*Monday, Tuesday, ..., Sunday*} giving the current day of the week.
- `Hour`: categorical variable with 24 configurations: {*0, 1, ..., 23*} giving the current hour.
- `Month`: categorical variable with 12 configurations: {*January, February, ..., December*}.

According to our available knowledge concerning the problem, these variables can be regrouped into simpler variables of interest. For example:

- The configurations of variable `DayOfWeek` can be grouped to form the variable `IsWeekEnd`, with configurations {*yes, no*}, respectively corresponding to the following groups {*Monday, ..., Friday*}, {*Saturday, Sunday*}.

- The configurations of variable `Hour` can be grouped to form the variable `PeriodOfTheDay`, with configurations {*Morning, LunchTime, Afternoon, Evening, Night*}, respectively corresponding to the following groups: {*7, 8,9,10,11*}, {*12, 13*}, {*14, 15, 16, 17, 18*}, {*19, 20, 21*} and {*22, 23, 1, 2, 3, 4, 5, 6, 7*}.

Such variables can be extracted from the time stamp alone, as long as (i) we possess a referential time for which we know the value of these variables and (ii) the rules of evolution of these variables through time is known.

In this thesis, we only consider categorical time information variables, as their inclusion into a HSDBN is convenient and often leads to interpretable models. In practice, such variables should be extracted from the time stamp in the dataset $D^{\mathbf{XY\tilde{Y}}}$, and considered in every algorithm as belonging to $\mathbf{X}$. Even though they are not technically metadata, and should be separated for more readability when displaying the structure, the accumulated experience from experiments suggests that the structural constraints concerning the nodes associated with these variables should be the same as the ones for $V_{\mathbf{X}}$.
As a rule of thumb: the variables we generally decide to extract are: `DayOfWeek`, `Week`, `DayOfYear`, `Year`, `DayOfMonth`, `Month` and `Hour`. Each of those variables may be reduced to more simple ones, as the `IsWeekEnd` or `PeriodOfTheDay`, depending on our prior knowledge and on the application. In Chapter 4, we will discuss a method to automatically reduce categorical variables into variables with less configurations.

3.3.2 Example

We consider the setting of the example presented in Section 1.1.3. The graph displayed in Figure 3.4 is an example of HSDBN jointly modeling the static variables $\mathbf{X}$ and the temporal variables $\mathbf{Y}$. In this updated example of HSDBN, we include the time information variable `IsWeekEnd`, denoted by W.[12]

Compared to the structure presented in Figure 3.2, the only difference is that the variable Y_3 now has two parent variables: W (`IsWeekEnd`) and X_2 (`Zone`).

This implies a new local conditional distribution for Y_3, which will now be modeled as a mixture of Gaussian variables with one component per configuration of (W, X_2), *i.e.* twice more than it was presented in Figure 3.2. Table 3.9 presents an example of a set of parameters defining the distribution $Y_3|X_2, W$.

In this example table, we see that variable W has a significant importance on the distribution of Y_3, which of course is intuitive. This illustrates how time information variables can be useful

[12]Note that in this context, many other time information variables could probably have an important added value, notably `Hour` and `Month`, but we only include `IsWeekend` in the example for the sake of illustration.

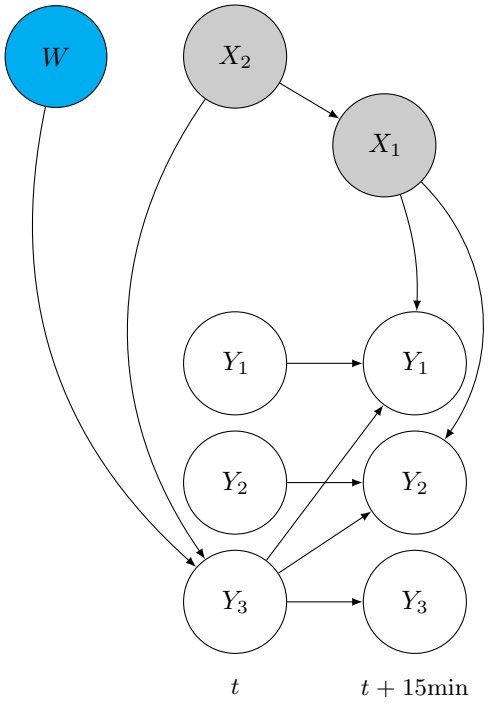


Figure 3.4: Example structure of a hybrid static dynamic Bayesian network in the setting of example ($\star$). Static nodes are colored in light gray, temporal nodes in white, and the node corresponding to a time information (`IsWeekEnd`) in blue.

in a model describing a temporal process. This is all the more true for HSDBN models, which inherit the flexibility of Bayesian networks in terms of continuous and categorical data joint modeling capabilities.

4 Concluding remarks and ongoing work

The formalism and associated algorithms introduced in this chapter enable us to use hybrid static-dynamic Bayesian networks to model temporal and static data from the real-world. Thanks to the proven efficiency of the `ds-BNSL` algorithm presented in Chapter 2, the HSDBN structure learning algorithm `HybridStaticDynamicBNSL` presented in Section 2.2 runs in a reasonable computing time, and some very promising preliminary results have been obtained on datasets from the `HOMES` programme. These results demonstrate the potential of HSDBN models, both in terms of interpretability and ability to answer queries, in particular metadata recovery from temporal observations using Algorithms 10 and 11 presented in Section 2.1).

Some ongoing experiments aim at properly comparing the two inference algorithms in diverse contexts, as well as providing results on other cross-field applications such as forecasting or critical event dependency analysis, using data from Schneider Electric.

x_2	w	μ_{x_2}	σ_{x_2}
Perch1	*no*	2.1	0.4
Perch2	*no*	3.0	0.6
Perch3	*no*	1.7	0.4
MeetRoomEast	*no*	4.1	1.9
MeetRoomWest	*no*	1.5	0.7
Box	*no*	0.8	0.4
Perch1	*yes*	0.2	0.2
Perch2	*yes*	1.0	0.8
Perch3	*yes*	0.1	0.1
MeetRoomEast	*yes*	0.1	0.1
MeetRoomWest	*yes*	0.5	0.2
Box	*yes*	0.0	0.1

Table 3.9: Example of a set of parameters $\Theta_3^{\mathbf{Y}} = \{(\mu_{x_2}, \sigma_{x_2})\}_{x_2 \in Val(X_2)}$, defining the distribution $Y_3|X_2 = x_2, W = w \sim \mathcal{N}(\mu_{x_2,w}, \sigma^2_{x_2,w})$

Chapter 4

Discussion and perspectives

Contents

In this chapter, we first discuss the evaluation of the generalization performance of Bayesian networks with the validation log-likelihood score, and give insights on how this score prevents overfitting (Section 1). We then propose a post-processing algorithm performing local operations on the graphs returned by `(q)ds-BNSL` to decrease the models complexity, and we give the idea of a criterion to guide the choice of the hyperparameter ε for the `QuasiDeterScreen` (Section 2). Finally, we present questions that remain to be answered concerning extensions of propositions presented in Chapter 2 to the BIC score, as well as links that can be drawn with the original Chow&Liu algorithm (Section 3).

1 Evaluation of Bayesian networks with the VLL score

In the experiments conducted in Section 5 of Chapter 2, we use two different ways to evaluate the performance of Bayesian networks: the BDe and (C)VLL scores. As opposed to the former, the VLL score is barely mentioned in the Bayesian network literature. In this section, we adopt an information theoretic point of view on parameter learning, and use this approach to shed light on the VLL score, and give insights on how it prevents overfitting. These insights are then illustrated by a simple example, and we conclude by presenting pending questions and prospective results.
The proofs of all proposed Lemmas and Propositions are available in Appendix A.3.

1.1 Introduction and notations

1.1.1 Notions of information theory

Cross-entropy and Kullback-Leibler divergence Suppose X is a categorical random variable, and let $p, q \in \mathcal{F}^P(Val(X))$ be two functions that define possible distributions for X,

- The cross-entropy of p with respect to q (asymmetrical expression) is defined as:

$$\mathcal{H}(p||q) = -\sum_{x \in Val(X)} p(x) \log(q(x))$$

  We denote $\mathcal{H}(p||p) = \mathcal{H}(p)$, the entropy of distribution p.[1]

- The Kullback-Leibler divergence from q to p (Kullback and Leibler, 1951) is defined as:

$$D_{KL}(p||q) = \sum_{x \in Val(X)} p(x) \log\left(\frac{p(x)}{q(x)}\right).$$

  Note that we have the relation:

$$D_{KL}(p||q) = \mathcal{H}(p) - \mathcal{H}(p||q). \tag{4.1}$$

[1] We use the notation $\mathcal{H}$ to denote the entropy and cross entropy of distribution(s) (for fixed variable(s)), and H to denote the entropy of random variable(s) (with a fixed distribution(s)).
Therefore if $p \in \mathcal{F}^P(Val(X))$ is the distribution of X, we have that $\mathcal{H}(p) = H(X)$.

Note that Jensen's inequality gives:

$$\forall q \in \mathcal{F}^P(Val(X)),\ \mathcal{H}(p||q) \leq \mathcal{H}(q),$$

which proves that the Kullback-Leibler divergence is nonnegative.

Both the cross entropy $\mathcal{H}(.||.)$ and the Kullback-Leibler divergence are in some ways measures of (asymmetric) dissimilarity from one distribution to another.

1.1.2 Setting: random variables, datasets and distributions

In the rest of this section, we suppose that D is a dataset containing M observations of the tuple of categorical variables $\mathbf{X} = (X_1, \ldots, X_n)$. This dataset is randomly split in a **training** set T and a **validation** set V (fixed from now on), such that $T \sqcup V = D$.
M_T, M_V, $C^T(.)$, $C^V(.)$ respectively denote the number of rows of T and V, and the associated count functions.

For any $p \in \mathcal{F}^P(Val(\mathbf{X}))$, we extend the notation $p(.)$ to any subset of variables in $\mathbf{X}$, as explained in Chapter 1.
Moreover, we denote by $p^* \in \mathcal{F}^P(Val(\mathbf{X}))$ the (true) underlying distribution of $\mathbf{X}$.

Finally, for a given graphical structure $G \in DAG_{[\![1,n]\!]}$ and

$$\Theta = \left\{ \{\theta_{x_i|\mathbf{x}_{\pi^G(i)}}\}_{x_i, \mathbf{x}_{\pi^G(i)}} \right\}_i \in \vartheta_G\ ,$$

we denote by p_Θ the distribution of $\mathbf{X}$ parametrized by Θ, such that for all $i \in [\![1,n]\!]$ and $(x_i, \mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi^G(i)})$,

$$p_\Theta(x_i|\mathbf{x}_{\pi^G(i)}) = \theta_{x_i|\mathbf{x}_{\pi(i)}},$$

and such that p_Θ factorizes in G, *i.e.* $\forall (x_1, \ldots, x_n) \in Val(\mathbf{X})$,

$$\begin{aligned} p_\Theta(x_1, \ldots, x_n) &= \prod_{i=1}^{n} p_\Theta(x_i|\mathbf{x}_{\pi(i)}) \\ &= \prod_{i=1}^{n} \theta_{x_i|\mathbf{x}_{\pi(i)}}. \end{aligned}$$

1.1.3 Why be interested in the VLL score ?

As presented before, inspired notably by Koller and Friedman (2009), Bayesian networks are used for two main reasons:

- knowledge discovery (that is also referred to as *interpretability* or *qualitative performance*),

- density estimation (that is also referred to as *quantitative performance*).

As far as density estimation is concerned, what we are seeking is good generalization capability: how well the distribution encoded in the Bayesian network fits unseen data ? Intuitively, this information should be measured by the VLL score (presented in Section 3.4.2 of Chapter 1), which is inspired from the train-validation procedure that is very common in supervised learning (see for example Friedman et al. (2001)).

In this section, we are interested in studying how well the VLL score, is effectively measuring the generalization performance of Bayesian networks, and notably how it naturally penalizes overly complex graphs.

1.1.4 Overfitting and Bayesian networks

By definition, we are expecting the VLL score of a Bayesian network to capture its generalization performance. This notably implies that it prevents overfitting: the score should decrease when the BN structure becomes overly complex, instead of consistently increasing with complexity like the MLL score.
For that purpose, let us first grasp intuitively why the MLL score leads to overfitting in the case of Bayesian network structure learning.

Spurious arc in a Bayesian network We consider a tuple of categorical variables $\mathbf{X} = (X_1, \dots, X_n)$, observed in a dataset D split into a training set T and a validation set V. We consider $G = (\llbracket 1, n \rrbracket, A) \in DAG_{\llbracket 1,n \rrbracket}$, and suppose there exist $k, l \in \llbracket 1, \dots, n \rrbracket$ two distinct integers such that $(k, l) \notin A$ and $G' = (\llbracket 1, n \rrbracket, A \sqcup (k, l))$ is still a DAG. In other words, G' is obtained by adding the arc (k, l) to the DAG G.
Finally, we suppose that G is a perfect map for the distribution P of $\mathbf{X}$. The arc (k, l) present in the graph G' is therefore spurious: it should not be learned by an ideal structure learning algorithm.

The fact that G is a perfect map for the distribution $P(\mathbf{X})$ implies that X_l is independent ($\perp_P$) of its non descendants given its parents (in G). Since k cannot be a descendent of l in G (otherwise G' would contain a cycle), we notably have:

$$X_l \perp_P X_k | \mathbf{X}_{\pi^G(l)}, \tag{4.2}$$

which implies:

$$I(X_l, X_k | \mathbf{X}_{\pi^G(l)}) = 0,$$

where I denotes the mutual information.

MLL score and spurious arcs From Lemma 3 stated in Chapter 1, we know that adding the (spurious) arc (k, l) to the structure G increases its MLL score by $I^D(X_l, X_k|\mathbf{X}_{\pi^G(l)})$. However, $I^D(X_l, X_k|\mathbf{X}_{\pi^G(l)})$ is the estimator of $I(X_l, X_k|\mathbf{X}_{\pi^G(l)})$ based on the empirical distribution p^D, and has very little chance of being exactly 0, despite the fact that $I(X_l, X_k|\mathbf{X}_{\pi^G(l)}) = 0$ holds: a MLL score-based approach would not reject this arc.
More generally, this explains why MLL score-based approaches add arcs which do not have a statistically sound contribution. As mentioned in Chapter 1, this problem is mostly tackled by penalization of the likelihood, whether it is explicit (BIC, AIC), or implicit (BD scores).

Remark Note that this motivates the use of independence tests, where given a type 1 error rate α, we compute a threshold $t_\alpha > 0$ from the asymptotic distribution of $I^D(X_l, X_k|\mathbf{X}_{\pi^G(l)})$ such that the hypothesis defined by Equation (4.2) is not rejected iff $I^D(X_l, X_k|\mathbf{X}_{\pi^G(l)}) \leq t_\alpha$. However, purely score-based approach do not imply such tests, even though some work was done towards including independence test information into Bayesian network scores (de Campos, 2006).

In the next subsection, we first give an information theoretic approach on the maximum likelihood estimation problem, enabling to get a better grasp on the notions of cross entropy and Kullback-Leibler divergence.

1.2 Another approach on the maximum likelihood estimation problem

We now rewrite the general maximum likelihood estimation (MLE) approach presented in Chapter 1 as a cross-entropy maximization problem, allowing the notions of cross entropy and Kullback-Leibler divergence to be seen in a familiar context. This is well known and the reader can for example refer to Murphy (2012) for further detail.

1.2.1 Minimizing the KL divergence from the real distribution p^*

We suppose we have a fixed DAG structure G. The problem of learning the associated parameters Θ, such that (G, Θ) is a Bayesian network, is classically solved with the MLE approach presented in Section 3.1 of Chapter 1.

It can also be tackled as an optimization problem: we wish to approach as close as possible p^*, the real distribution of $\mathbf{X}$, with a distribution that factorizes in G.
Using the KL-divergence as a measure of dissimilarity from a distribution to another, our problem narrows down to minimizing the KL-divergence from p_Θ to p^*, for all $\Theta \in \vartheta_G$. The distribution we are looking for is therefore defined by $p_{\hat{\Theta}^{KL}}$, where $\hat{\Theta}^{KL}$ satisfies:

$$\hat{\Theta}^{KL} \in \underset{\Theta \in \vartheta_G}{\operatorname{argmin}} D_{KL}(p^*||p_\Theta). \tag{4.3}$$

1.2.2 Approximating the true distribution p^* using the empirical distribution p^D

In practice, we do not know the underlying distribution p^* that we wish to approach. A natural option is to approximate it with the empirical distribution p^D, defined as:

$$\forall I \subset [\![1,n]\!] \text{ and } \mathbf{x}_I \in Val(\mathbf{X}_I),\ p^D(\mathbf{x}_I) = \frac{C^D(\mathbf{x}_I)}{M}.$$

Problem stated in Equation (4.3) is therefore simplified as the following proxy optimization problem:

$$\hat{\Theta}^{KL,D} \in \underset{\Theta \in \vartheta_G}{\operatorname{argmin}}\ D_{KL}(p^D||p_\Theta). \tag{4.4}$$

Using Equation (4.1) and the fact that $\mathcal{H}(p^D)$ does not depend on Θ, we remark that $\hat{\Theta}^{KL,D}$ defined in Equation (4.4) also satisfies:

$$\hat{\Theta}^{KL,D} \in \underset{\Theta \in \vartheta_G}{\operatorname{argmin}}\ \mathcal{H}(p^D||p_\Theta). \tag{4.5}$$

1.2.3 Linking these optimization problems to log-likelihood maximization

We first show the following lemma, which is a preliminary result enabling to decompose the distributions in expressions given in Equation (4.4) or Equation (4.5).

Lemma 9 *Let $G \in DAG_{[\![1,n]\!]}$ be a Bayesian network structure modeling $\mathbf{X}$. Then for any given set of parameters $\Theta \in \vartheta_G$, we have:*

$$\mathcal{H}(p^*||p_\Theta) = \sum_{i=1}^{n} \sum_{x_i, \mathbf{x}_{\pi(i)}} p^*(x_i, \mathbf{x}_{\pi(i)}) \log(p_\Theta(x_i|\mathbf{x}_{\pi(i)})). \tag{4.6}$$

Remark In intuitive terms, this result states that the second distribution in the cross entropy formula 'imposes' its decomposition to the first distribution.

Lemma 9 enables us to show the following proposition, that formally links the optimization problem given in Equation (4.4) to the usual MLE problem presented in Equation (1.7).

Proposition 9 *Under the previously defined notations,*

$$\underset{\Theta \in \vartheta_G}{argmin}\ D_{KL}(p^D||p_\Theta) = \underset{\Theta \in \vartheta_G}{argmax}\ l_D(\Theta).$$

This gives another lighting on using MLE for learning parameters corresponding to a fixed Bayesian network structure G: it is equivalent to finding the parameter $\Theta \in \vartheta_G$ such that the Kullback-Leibler divergence from p^D (natural approximation of the real distribution p^*) to p_Θ is minimal.

This result is stated in a somewhat similar manner in Koller and Friedman (2009) and Murphy (2012).
Now that the gap between notions such as cross entropy or KL-divergence and the log-likelihood is bridged, we are interested in using this setting to shed light on the VLL score from an information theoretic perspective.

1.3 Validation log-likelihood score: a new perspective

In this subsection, we show how the validation log-likelihood score can be expressed using information theoretic tools. We then explain how this point of view enables interesting insights in terms of how the VLL score evaluates Bayesian networks quantitative performance.

We still consider a structure $G \in DAG_{[\![1,n]\!]}$, corresponding to a Bayesian network modeling $\mathbf{X} = (X_1, \dots, X_n)$, as well as a dataset D (randomly split in $T \cup V$) containing M observations of $\mathbf{X}$.

1.3.1 Rewriting the validation log-likelihood score using cross entropy

In Lemma 10, we show how the VLL score, presented in Equation (1.13), can be rewritten using the notion of cross entropy.

We remind that in the present setting, $\hat{\Theta}^D$ defined in Equation (1.7) denotes the maximum likelihood estimate of the parameter (set) $\Theta \in \vartheta_G$ given D (which we now know also minimizes the KL-divergence from p_Θ to p^D).
$\hat{\Theta}^D$ defines all the local conditional distributions $X_i|\mathbf{X}_{\pi(i)}$ for $i \in [\![1, n]\!]$, *i.e.*

$$\hat{\Theta}^D = \{\hat{\theta}^D_{x_i|\mathbf{x}_{\pi^G(i)}} \mid (x_i, \mathbf{x}_{\pi^G(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi^G(i)}),\ 1 \leq i \leq n\}.$$

Lemma 10 *If $\hat{\Theta}^T$ and $\hat{\Theta}^V$ are the maximum likelihood estimates of $\Theta \in \vartheta_G$ with respect to datasets T and V respectively, the VLL score of G w.r.t. T, V defined in Equation (1.13) has the following expression:*

$$s^{VLL}_{T,V}(G) = -M_V \mathcal{H}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T}). \tag{4.7}$$

Note that the MLL score, defined by $s^{MLL}_D(G) = l_D(\hat{\Theta}^D)$, can be rewritten in a very similar way:

$$s^{MLL}_D(G) = -M\mathcal{H}(p_{\hat{\Theta}^D} || p_{\hat{\Theta}^D}) = -M\mathcal{H}(p_{\hat{\Theta}^D}). \tag{4.8}$$

This result is interesting and gives some interesting information, but we rather choose to use it to prove an even more insightful property, stated in Proposition 10.

We denote by $\bar{s}$ the per-sample normalized version of a score s. For instance, for $G \in DAG_{[\![1,n]\!]}$, we have:

$$\bar{s}_D^{MLL}(G) = \frac{s_D^{MLL}(G)}{M},$$
$$\bar{s}_{T,V}^{VLL}(G) = \frac{s_{T,V}^{VLL}(G)}{M_V}.$$

We now state the main proposition of this section, which extends Lemma 10.

Proposition 10 *If $\hat{\Theta}^T$ and $\hat{\Theta}^V$ are the maximum likelihood estimates of $\Theta \in \vartheta_G$ with respect to datasets T and V respectively, the per-sample normalized VLL score of G w.r.t. T, V has the following expression:*

$$\bar{s}_{T,V}^{VLL}(G) = \bar{s}_V^{MLL}(G) - D_{KL}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T}). \tag{4.9}$$

Equation (4.9) is particularly interesting because it links the VLL score for Bayesian networks to a concept that is common in the supervized learning context: the bias-variance tradeoff.

- The term $\bar{s}_V^{MLL}(G)$ can be interpreted as the **bias** term: it measures how well the structure fits the data. This term alone would lead to learn complete graphs, since we showed it can only grow with the number of arcs of the structure G.

- The term $D_{KL}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T})$ represents the **variance** of the model that is learned (implied by G), and grows with the complexity of the structure: it captures how robustly the parameters were learned on T, by comparing them to the parameters learned on an unseen dataset (V) generated by the same underlying distribution.
  The more complex the learned structure is, the less robustly the parameters are learned, and the bigger $D_{KL}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T})$ gets.

An spurious will most certainly have a positive contribution to the term $\bar{s}_V^{MLL}(G)$, as explained in Section 1.1.4. However, we can expect it to have a negative overall contribution to the VLL score, since the parameters learned relative to this arc only capture noise, which should result in an important contribution to the $D_{KL}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T})$ term.

Remark: caution when using the VLL score Using the VLL score to evaluate a structure must be done with caution: this is only justified for a structure that has been proposed / learned independently of V ! An expert that gets inspiration from the entire dataset D to propose a structure, then uses the VLL score to evaluate it makes an important mistake.
In practice, the VLL score should be used either to score a graph that was proposed independently from the data, or to evaluate a structure learning algorithm (that we run only on dataset T), but not the associated structure.

1.4 Experiments: study of the VLL score on a simple example

In this subsection, we present an example illustrating how the VLL score effectively prevents overfitting, and enables to find optimal fit-complexity tradeoff.

1.4.1 Description of the example

We consider a set of gardens that possess automatic sprinklers, and suppose we have a number of *i.i.d* observations of 5 categorical variables describing such gardens:

- `Season` (D): *Winter* or *Summer*
- `Weather` (W): *Sunny* or *Rainy*
- `Sprinkler` (S): *On* or *Off*
- `Grass` (G): *Wet* or *Dry*
- `HouseColor` (H): *White*, *Brown*, *Black* or *Yellow*

We suppose the graph G^*, displayed in Figure 4.1, is a perfect map for the distribution P of those 5 variables.

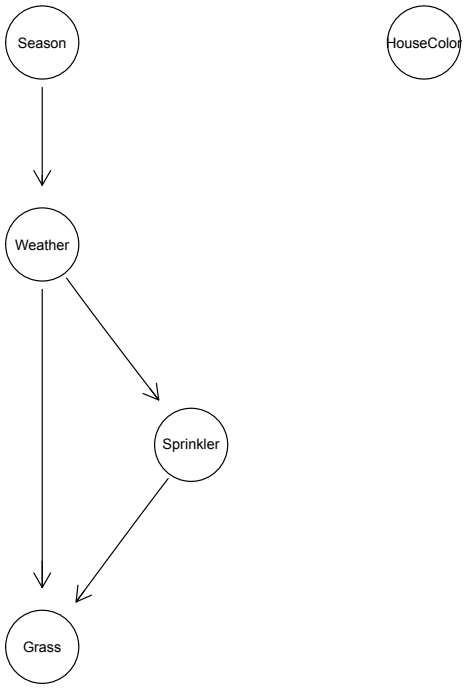


Figure 4.1: Example Bayesian network structure G^*

Therefore, $P(H, D, W, S, G)$ decomposes as follows:

$$P(H, D, W, S, G) = P(H)P(D)P(W|D)P(S|W)P(G|W, S). \tag{4.10}$$

The parameters Θ^* presented in Table 4.1 define the Bayesian network $\mathcal{B}^* = (G^*, \Theta^*)$, which rightfully models the distribution P of (H, D, W, S, G).

$h \in Val(H)$	*white*	*brown*	*black*	*yellow*
θ_h	0.2	0.5	0.2	0.1

(a) `HouseColor`

$d \in Val(D)$	*winter*	*summer*
θ_d	0.5	0.5

(b) `Date`

$w \in Val(W)$	*sunny*	*rainy*
$\theta_{w\|D=winter}$	0.2	0.8
$\theta_{w\|D=summer}$	0.7	0.3

(c) `Weather`|`Season`

$s \in Val(S)$	*on*	*off*
$\theta_{s\|W=sunny}$	0.7	0.3
$\theta_{s\|W=rainy}$	0.2	0.8

(d) `Sprinkler`|`Weather`

$g \in Val(G)$	*wet*	*dry*
$\theta_{g\|S=on,W=sunny}$	0.5	0.5
$\theta_{g\|S=on,W=rainy}$	0.9	0.1
$\theta_{g\|S=off,W=sunny}$	0.1	0.9
$\theta_{g\|S=off,W=rainy}$	0.7	0.3

(e) `Grass`|{`Sprinkler`, `Weather`}

Table 4.1: Conditional Probability Tables defining parameters Θ^*, associated with the structure G^*

1.4.2 Experiment description

Now that we have defined the graph structure G^* and the associated parameters Θ^*, we are able to generate data from the Bayesian network $\mathcal{B}^* = (G^*, \Theta^*)$.

Here is the protocol we use in order to test the VLL score as a model selection criteria.

- We generate $M = 500$ observations of the 5 considered variables.[2]
- We consider several different structures:
  1. We **manually select** graphs that are close to G^* (in terms of elementary operations on graphs[3]), both by discarding arcs and adding arcs to G^*.

[2]The fact that VLL score measures cross entropy from $p_{\hat{\Theta}^T}$ to $p_{\hat{\Theta}^T}$ suggests that the spurious arcs that will be the most penalized are those for which parameters are not learned robustly. In the limit of big data, it is less of a problem to learn a spurious arc since even parameters corresponding to spurious arcs are robustly learned, (even if they encode the fact that the corresponding arc is not necessary). Therefore, even if the arc is not interpretable in a qualitative sense, learning it does not cause any issue in terms of density estimation (although it slows down inference). In the spirit of the example, we choose a reasonably small number of observations in order to allow overfitting to happen.

[3]Arc addition, removal and reversal.

2. We **learn graphs** by running two standard structure learning algorithms.

- Finally **compute the MLL score** (training performance) and **the VLL score** (generalization performance) for all of the structures we manually selected and learned. We are expecting an increasing curve with respect to structure complexity in the first case, and an increasing then decreasing curve in the second case, thus clearly expressing the tradeoff.

1.4.3 Results: learned graphs

The learned graphs are learned using two standard structure learning algorithms:

- `HillClimbing`, presented in Algorithm 2 in Section 3.2.2 of Chapter 1 (with 10 random restarts and a length 10 tabu list).
- `MaxMinHillClimbing`, which is an hybrid structure learning algorithm proposed by Tsamardinos et al. (2006), which contains two phases:
  - a *constraint-based phase* where the structure space is restricted to a graph skeleton containing allowed edges, (using mutual information independence tests and first order error threshold $\alpha = 0.05$),
  - a *score&search phase* using `HillClimbing` (with the same settings) to go through the restricted structure space.

Both of these algorithms are used with the $BIC(\lambda)$ score (defined in Section 1) as a target[4], for $\lambda \in \{0, 0.1, 0.25, 1, 10, 30\}$[5].
This enables us to test the VLL score on graphs structures that are realistically returned by structure learning algorithms. Moreover, this gives us an insight about which algorithms / score combination is the closest to finding the rightful BN.

Figure 4.2 and Figure 4.3 display a representative extract of the structures learned by these algorithms.

We remark that none of these algorithms recover the exact structure G^*. However, the graph that is learned by `HillClimbing` with $BIC(\lambda = 1)$, and by `MaxMinHillClimbing` with $BIC(\lambda)$ for $\lambda \in \{0, 0.1, 1\}$ (respectively displayed in Figure 4.2 (c) and Figure 4.3(a),(b),(c)) is however very close (one arc reversal) to G^*, and has the exact same complexity (in terms of number of parameters). We expect this graph to have the best VLL score, since it is the 'closest' to G^*

[4] These results were produced before we were familiar with the works by Silander et al. (2007) which explain how sparsity can be induced by decreasing the ESS of the BDe score used as a target in a score&search algorithm.

[5] We recall that $BIC(\lambda = 0)$ corresponds to the MLL score, and $BIC(\lambda = 1)$ to the usual BIC score. Testing different values of λ for the BIC score can be interpreted as artificially changing the size of the dataset D to simulate different structure learning settings with the same variables.

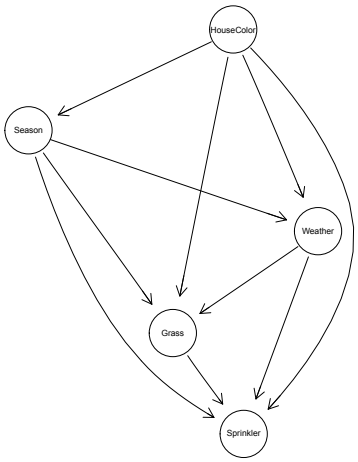
intuitively.

Figure 4.4 displays the evolution of the MLL scores for the different learned graphs.

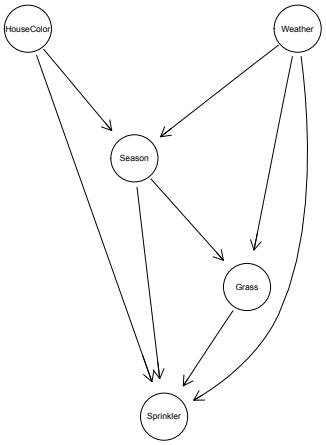
We see in Figure 4.4 that the MLL score increases with the structure's complexity (when λ decreases), and reaches its maximum for the graphs learned by the MLL score (the complete graph for the `HillClimbing` algorithm): this is overfitting.
In the case of the `MaxMinHillClimbing` algorithm, the structure space that is searched in the score&search phase is constrained by the first phase of the algorithm (called `MaxMinParentChildren`),which explains why the learned structures do not evolve when $\lambda < 10$: the densest graph in the allowed structure space is already reached. We however notice that spurious arcs add less to the MLL score than real arcs (*plateau* effect).

Figure 4.5 and 4.6 displays the evolution of the CVLL score (mean of the VLL score with 10 folds, computed 20 times with different random seeds) with respect to the parameter λ of the $BIC(\lambda)$ score used as a target for the `HillClimbing` and `MaxMinHillClimbing` algorithms. Figures 4.5(b) and 4.6(b) are zoomed versions of Figures 4.5(a) and 4.5(a) respectively.

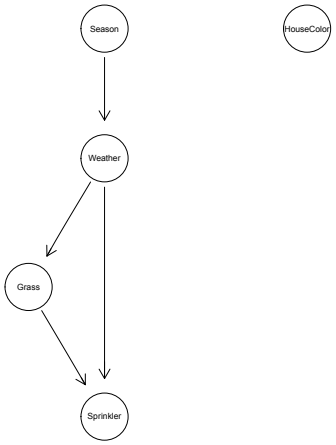
In Figure 4.5 and 4.6, we see that the CVLL score does a good job capturing the performance of structures: even though the difference is not very significant for structures that are very close to G^* (*e.g.* `HillClimbing` with $\lambda \in \{0.25, 1, 10\}$ or `MaxMinHillClimbing` with $\lambda < 10$), it becomes much greater when arcs that imply unnecessary complexity are added (as it is the case for graphs learned by `HillClimbing` with $\lambda \in \{0, 0.1\}$), or when important arcs are removed (graph learned by `MaxMinHillClimbing` with $\lambda = 30$).



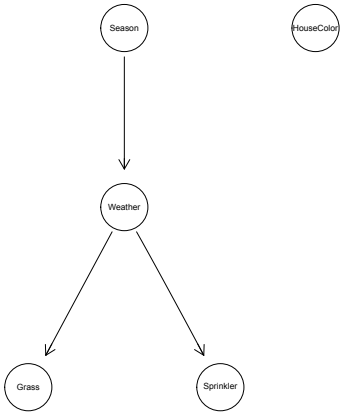
(a) Bayesian network structure learned by `HillClimbing` and MLL (BIC($\lambda = 0$)) as a target score



(b) Bayesian network structure learned by `HillClimbing` and BIC($\lambda = 0.1$) as a target score

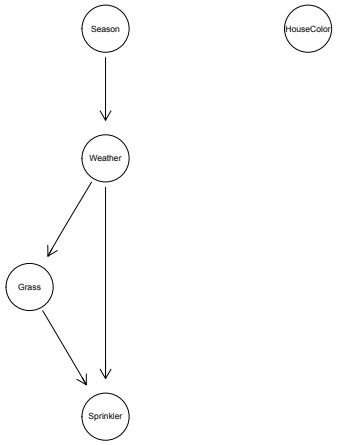


(c) Bayesian network structure learned by `HillClimbing` and BIC($\lambda = 1$) as a target score

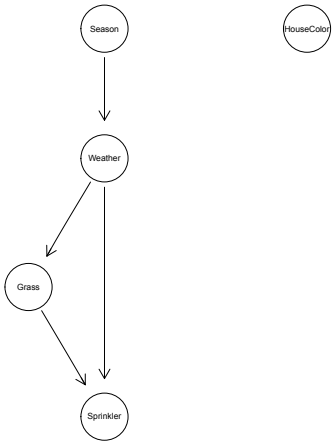


(d) Bayesian network structure learned by `HillClimbing` and BIC($\lambda = 30$) as a target score

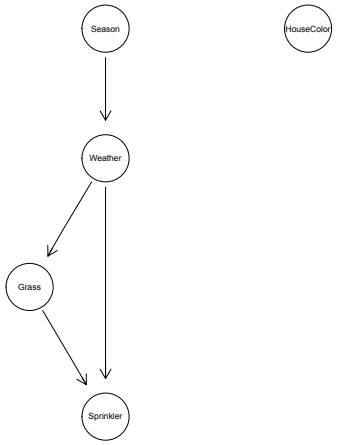
Figure 4.2: Bayesian network structures learned by the `HillClimbing` algorithm with different λ parameters for the BIC score



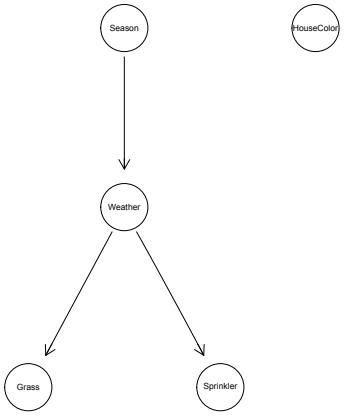
(a) Bayesian network structure learned by `MaxMinHillClimbing` and MLL (BIC($\lambda = 0$)) as a target score



(b) Bayesian network structure learned by `MaxMinHillClimbing` and BIC($\lambda = 0.1$) as a target score



(c) Bayesian network structure learned by `MaxMinHillClimbing` and BIC($\lambda = 1$) as a target score



(d) Bayesian network structure learned by `MaxMinHillClimbing` and BIC($\lambda = 30$) as a target score

Figure 4.3: Bayesian network structures learned by the `MaxMinHillClimbing` algorithm with different λ parameters for the BIC score

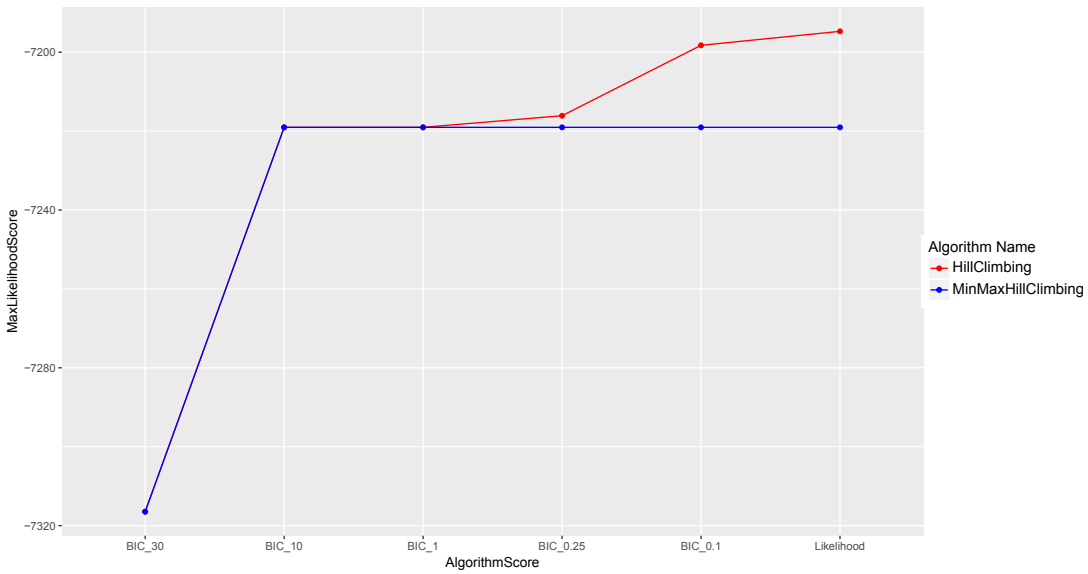
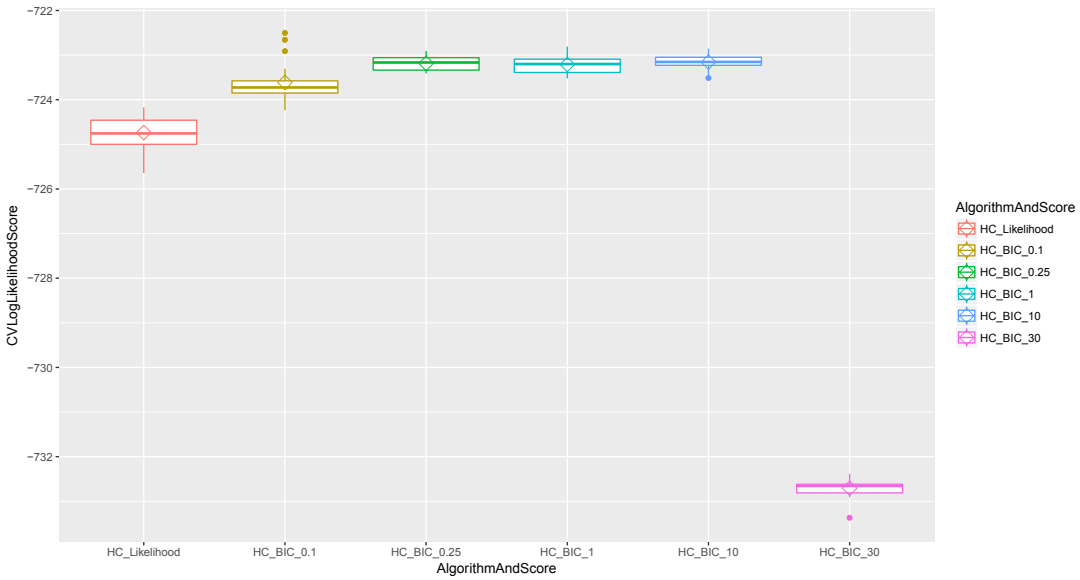
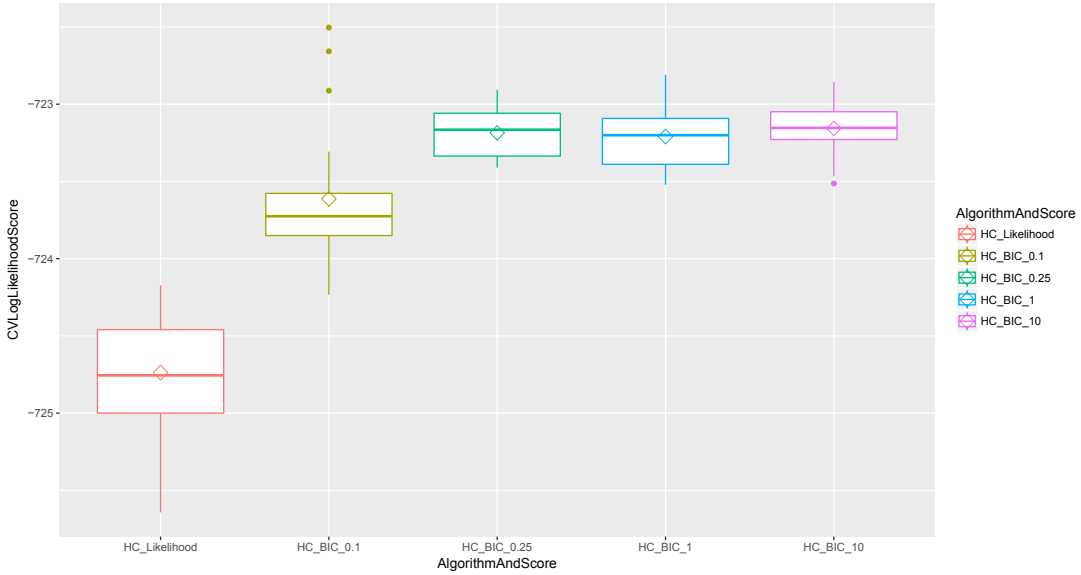


Figure 4.4: Evolution of the MLL score for structures learned by `MaxMinHillClimbing` and `HillClimbing` with the $BIC(\lambda)$ for different values of λ

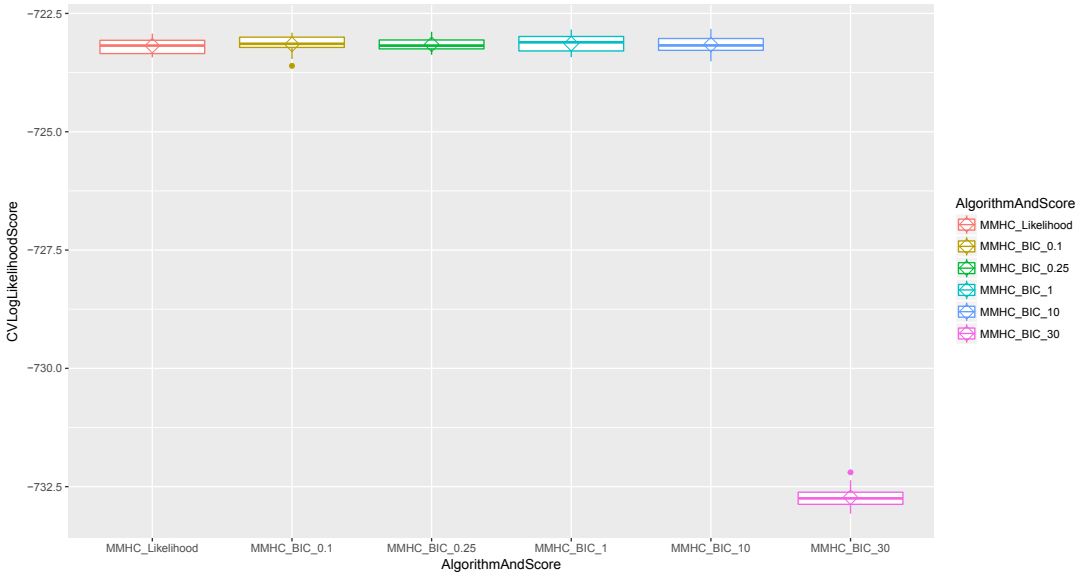


(a) `HillClimbing` structure learning algorithm

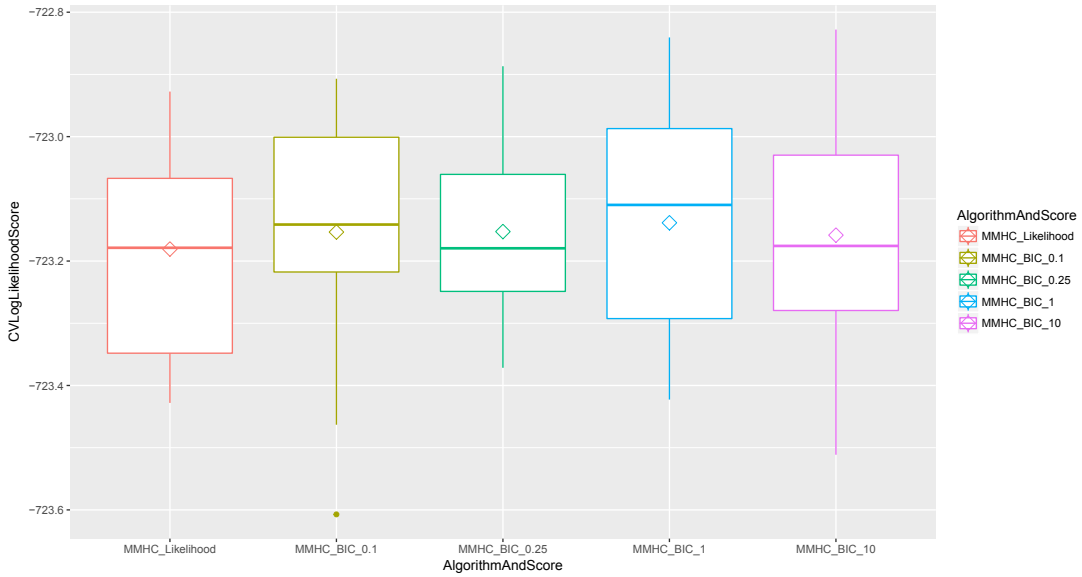


(b) `HillClimbing` structure learning algorithm (zoom)

Figure 4.5: Evolution of the CVLL score (10 folds and 20 runs) with respect to the λ parameter of the $BIC(\lambda)$ score used as a target in the `HillClimbing` structure learning algorithms



(a) `MaxMinHillClimbing` structure learning algorithm



(b) `MaxMinHillClimbing` structure learning algorithm (zoom)

Figure 4.6: Evolution of the CVLL score (10 folds and 20 runs) with respect to the λ parameter of the $BIC(\lambda)$ score used as a target in the `MaxMinHillClimbing` structure learning algorithms

1.4.4 Results: manually selected graphs

Here is a list of the abbreviations used in the figures and the corresponding operations made on G^*.

- *-2wgdw*: 2 removed arcs (`Weather`, `Grass`) and (`Season`, `Weather`),
- *-2wssg*: 2 removed arcs (`Weather`, `Sprinkler`) and (`Sprinkler`, `Grass`),
- *-1dw*: 1 removed arc (`Season`, `Weather`),
- *-1ws*: 1 removed arc (`Weather`, `Sprinkler`),
- *real*: no operation,
- *+1ds*: 1 added arc (`Season`, `Sprinkler`),
- *+1hd*: 1 added arc (`HouseColor`, `Season`),
- *+1dg*: 1 added arc (`Season`, `Grass`),
- *+1hg*: 1 added arc (`HouseColor`, `Grass`),
- *+2dwsg*: 2 added arcs (`Season`, `Sprinkler`) and (`Season`, `Grass`),
- *+2dwhd*: 2 added arcs (`Season`, `Sprinkler`) and (`HouseColor`, `Season`),
- *+2hshw*: 2 added arcs (`HouseColor`, `Sprinkler`) and (`HouseColor`, `Weather`),
- *+2hshg*: 2 added arcs (`HouseColor`, `Sprinkler`) and (`HouseColor`, `Grass`),
- *dense*: complete graph (respecting the ordering `HouseColor`, `Season`, `Weather`, `Sprinkler`, `Grass`).

Figure 4.7 display the evolution of the MLL scores for the different manually selected graphs.

In Figure 4.7, we see that the MLL score increases when we add spurious arcs to G^*. However, as it is observed for learned graphs, the difference in MLL score is much less important than when we remove real arcs.
Indeed, even if the MLL score continues to increase when the graph becomes more complex than G^*, we reach a plateau where this increase is very slow.

Figure 4.8 displays the CVLL score (mean of the VLL score with 10 folds, computed 20 times with different random seeds) for the manually selected structures.

In Figure 4.8, we see that the CVLL score is consistently better for the real structure G^* than for the other structures, however close in terms of elementary graphical operations. The differences are less important for graphs that contain spurious arcs compared to graphs that have missing arcs, although the differences are always consistent on the 20 runs that were made (as we see in the zoomed Figure 4.8(b)).

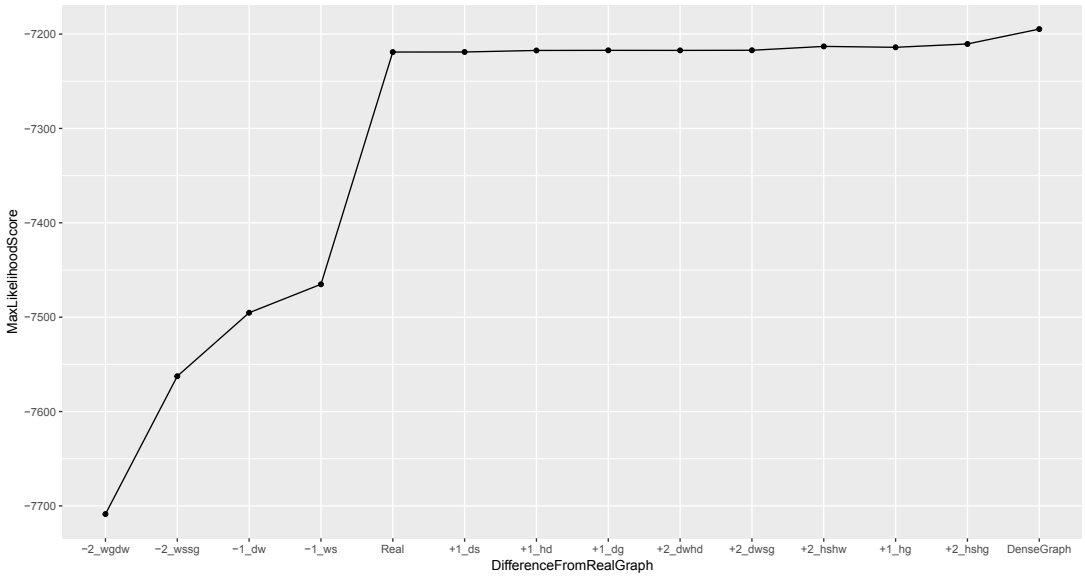
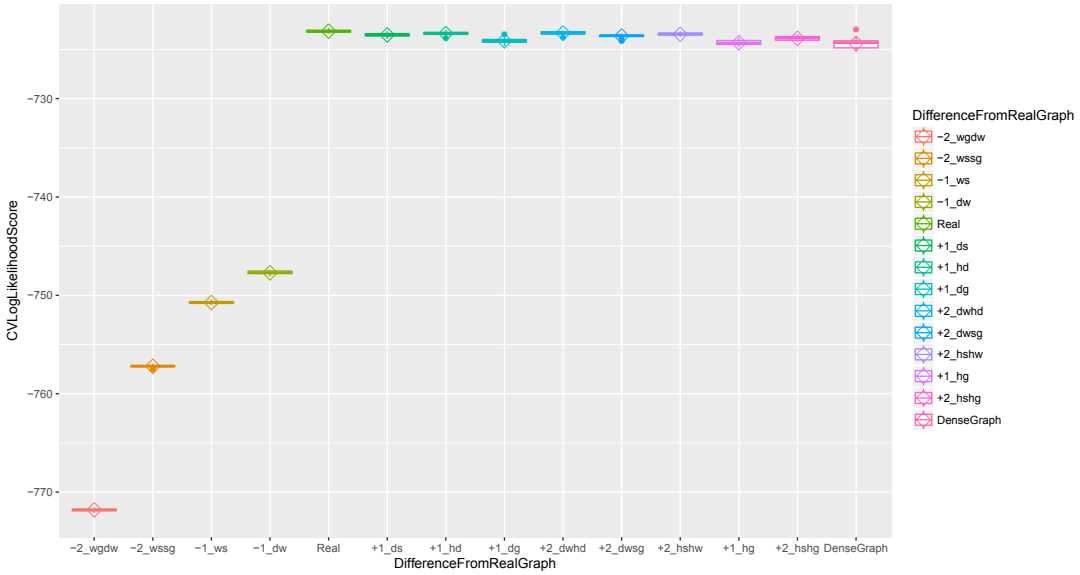
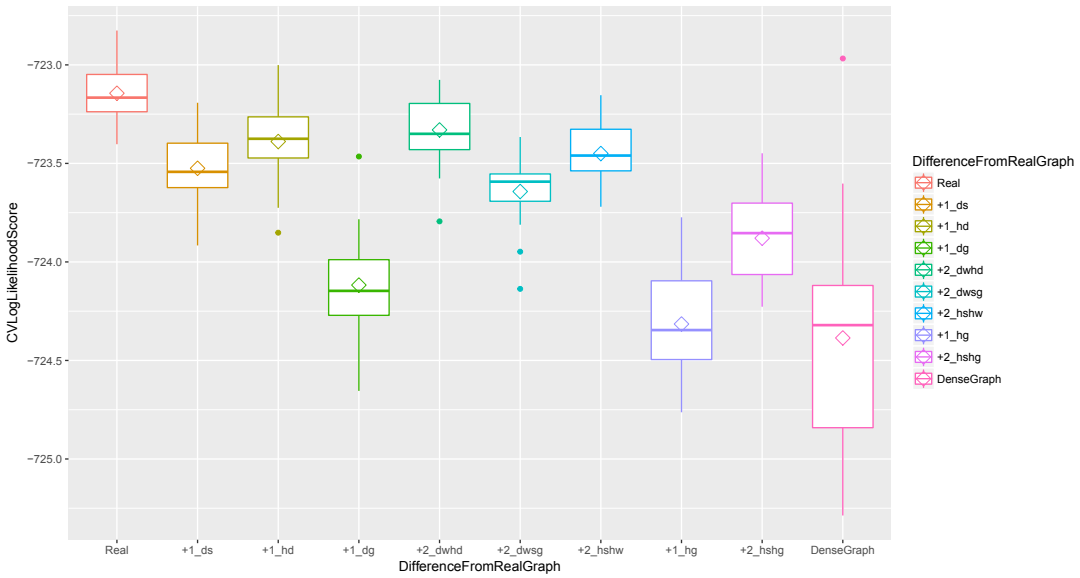


Figure 4.7: MLL scores for several selected structures around G^*



(a) CVLL score for several selected structures that are a few local operations away from the real structure G^*



(b) CVLL score for several selected structures that are a few local operations away from the real structure G^* (zoom)

Figure 4.8: CVLL score (10 folds and 20 runs) for several selected structures that are a few local operations away from the real structure G^*

1.4.5 Results summary

These results, although empirical, are very promising in terms of how well the VLL score captures the real underlying distribution, and notably how it effectively penalizes spurious arcs (thanks to the fact that parameters corresponding to those arcs are not robustly learned from data as suggested by Equation (4.9)).
Compared to widely used scores such as BIC and BDe, VLL has the advantage of being similar in construction to the MLL score. We believe that this could enable us to show a proposition analogous to Proposition 4 of Chapter 2, with respect to the VLL score. In the next subsection, we develop this idea, and present some questions that remain to be answered for such a result to be proven.

1.5 Extending theoretical results to the VLL score: food for thought

We are convinced that the VLL score indeed captures very interesting information (with regards to rightful structure recovery and to generalization performance). Considering that its expression has a similar structure to the one the MLL score, we would like to know if the result that we proved for the MLL score in Proposition 4, and that originally motivated the design of the `ds-BNSL` and `qds-BNSL` algorithms, still holds with respect to the VLL score.

We are confident that such a result could exist, even though it should imply stronger assumptions than in Proposition 4. This is notably motivated by the experiments conducted in Chapter 2, which reveal that structures learned by the `(q)ds-BNSL` algorithms often had very good VLL scores (even better than `sota-BNSL` in the case of `ds-BNSL`).

1.5.1 Graal result

We consider the same setting as in Chapter 2: we observe $\mathbf{X} = (X_1, \ldots, X_n)$ M times in a dataset D. F is a deterministic forest w.r.t. D.

The result we would like to prove is analogous to Proposition 4 with the VLL score instead of MLL.

Conjecture idea *Under a set of assumptions to be defined, if G_R maximizes the VLL score w.r.t. $D_{\mathcal{R}(F)}$, then*

$$G^* = F \cup G_R$$

maximizes the VLL score w.r.t. D.

Remark The VLL score $s^{VLL}_{T,V}$ in practice depends on the subsets T, V of D, and we should therefore be cautious with the notion of 'VLL maximization'. As mentioned in Section 3.4.2 of Chapter 1, if T and V are randomly sampled, the VLL score w.r.t. to D is denoted s^{VLL}_{D} and is a random variable. We could therefore consider the maximization property to be meant in expectancy, or holding under a (controlled) probability.

1.5.2 Elements of response

A key intermediate result in the proof of Proposition 4 is the fact that, for any $G_R \in DAG_{\mathcal{R}(F)}$,

$$s^{MLL}_{D}(G_R \cup F) = s^{MLL}_{D_{\mathcal{R}(F)}}(G_R).$$

If F is a deterministic forest with respect to D, a simple computation shows that for any $G_R \in DAG_{\mathcal{R}(F)}$,

$$s^{VLL}_{T,V}(G_R \cup F) = s^{VLL}_{T_{\mathcal{R}(F)},V_{\mathcal{R}(F)}}(G_R).$$

However this is not entirely satisfactory: the fact that F is a deterministic forest with respect to D implies that the entire dataset D has already been considered for building the DAG $G_R \cup F$, which invalidates the evaluation of this graph by $s^{VLL}_{T,V}$, as mentioned in the last subsection.
For a sound evaluation, F should be computed using the training set only: it would therefore be a deterministic forest w.r.t. T, which however does not imply a priori that it is deterministic with respect to D (although the converse is true). To guarantee this property, we would need a strong assumption such as:

Assumption idea *Empirical determinism is rare enough so that if T is a subset of D that is sufficiently representative of D (e.g. contains at least 50% of randomly sampled observations), relationships that are not deterministic w.r.t. D are not deterministic with respect to T neither.*

This assumption is reasonable in practice: in the experiments conducted on the metadatasets we used throughout this thesis, we have observed that the training sets T did not contain any spurious empirical deterministic relationships compared to D.
A simple property of the dataset D that should help our assumption to be satisfied is that the minimum of the set of positive empirical conditional entropies values, *i.e.*

$$Val_{H^D} = \{H^D(X_i|X_j), 1 \leq i, j \leq n\} \setminus \{0\},$$

is above a certain threshold.

Formally, we are looking for a number $a > 0$ (as big as possible) such that:

$$\min\left(\{H^D(X_i|X_j), 1 \leq i, j \leq n\} \setminus \{0\}\right) > a.$$

We expect $G_R \cup F$ (as defined in the conjecture) to maximize the VLL score w.r.t. D with a probability that is related to a. Ideally, there even might be a condition on a that guarantees that the result always hold.

2 Algorithmic perspectives

2.1 Decreasing the complexity of the ds-BNSL output using local search

2.1.1 Idea

Let us consider $\mathbf{X} = (X_1, \ldots, X_n)$ a tuple of n categorical random variables, D a dataset containing M observations of $\mathbf{X}$, and $F = \texttt{DeterScreen}(D)$ the deterministic forest w.r.t. D returned by the DeterScreen algorithm presented in Chapter 2. We remind that $\mathcal{R}(F) \subset [\![1, n]\!]$ is the set of root nodes of F.

In Proposition 4, we show that if G_R is a DAG that maximizes the MLL score w.r.t. to $D_{\mathcal{R}(F)}$, the DAG $F \cup G_R$ maximizes the MLL score w.r.t. D. This result led to the design of the ds-BNSL algorithm, which enables a substantial computational time gain for the Bayesian network structure learning task in presence of deterministic relationships, thanks to the fact that only a subset of the variables have to be considered by the (costly) structure learning algorithm sota-BNSL: the variables corresponding to the roots of F, $\mathcal{R}(F)$.
One may however wonder if the variables that are set aside by the screening phase, *i.e.* $\mathbf{X}_{[\![1,n]\!] \setminus \mathcal{R}(F)}$, could still be parents of other variables in the final graph (root variables only, since variables that already have a deterministic parent cannot benefit from another parent).

We recall that in practice we are rather looking to find a tradeoff between fit and complexity. With the DeterScreen algorithm (Algorithm 5 presented in Section 3 of Chapter 3) for instance, we are approaching this objective by minimizing the model complexity among structures which maximize the MLL score. Indeed, we propose a way to find the deterministic trees with a minimal number of parameters by choosing the less complex parents individually for each variables, using BestParent (Algorithm 4). This guarantees that the overall tree has minimal complexity among all deterministic trees (which all maximize the MLL score).

We wish to apply the same idea to the graph formed from the fusion of F and G_R. Indeed, in some cases, there exist local operations that can be applied to $F \cup G_R$, that decrease the total number of parameters of the structure while still guaranteeing the MLL-maximizing property. We call these **arc lowering** operations: they correspond to the simultaneous removal of an arc (r_1, r_2) with $r_1, r_2 \in \mathcal{R}(F)$ and the addition of an arc (v, r_2) with $v \in \texttt{Desc}^F(r_1)$.[6]

[6] $\texttt{Desc}^G(v)$ is the set of all the descendants of v in the DAG G.

To illustrate this, we now present a simple example.

2.1.2 Example of an arc lowering operation decreasing the number of parameters without loosing the MLL-maximization property

Presentation of the variables and associated assumptions We consider variables $\mathbf{X} = \{X_{11}, X_{12}, X_{13}, X_{14}, X_{15}, X_{21}, X_{22}, X_{23}\}$, observed in a dataset D such that:

$$\begin{aligned} H^D(X_{12}|X_{11}) &= 0, \\ H^D(X_{13}|X_{11}) &= 0, \\ H^D(X_{14}|X_{12}) &= 0, \\ H^D(X_{15}|X_{12}) &= 0, \\ H^D(X_{22}|X_{21}) &= 0, \\ H^D(X_{22}|X_{23}) &= 0. \end{aligned}$$

We suppose that no other deterministic dependencies hold[7] (notably there are no redundant variables in D).

Deterministic trees T_1 and T_2 We denote by v_{ij} the node corresponding to variable X_{ij} in an associated graphical structure. The deterministic forest F (w.r.t. D) returned by the `DeterScreen` algorithm can be written $F = T_1 \cup T_2$, where T_1 and T_2 are two deterministic trees with respect to D defined as:

$$\begin{aligned} T_1 &= (\{v_{11}, v_{12}, v_{13}, v_{14}, v_{15}\}, \{(v_{11}, v_{12}), (v_{11}, v_{13}), (v_{12}, v_{14}), (v_{12}, v_{15})\}), \\ T_2 &= (\{v_{21}, v_{22}, v_{23}\}, \{(v_{21}, v_{22}), (v_{21}, v_{23})\}). \end{aligned}$$

and is displayed in Figure 4.9.

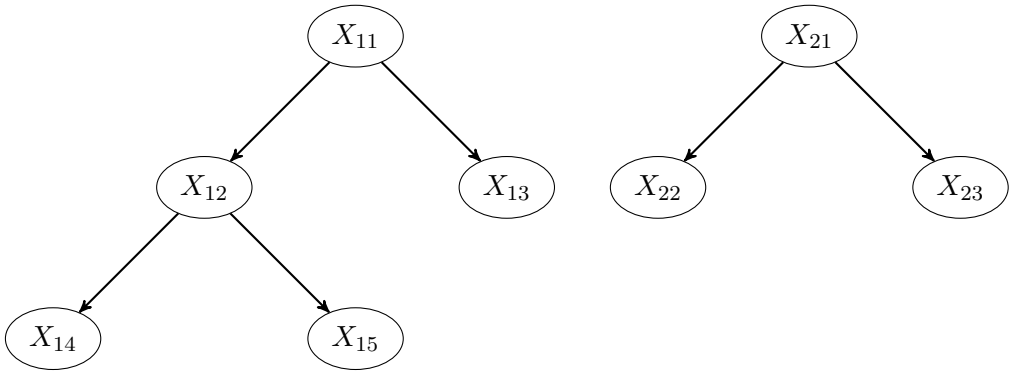


Figure 4.9: Deterministic forest F

[7] except for the ones that are directly implied by those previously presented, such as $H^D(X_{14}|X_{11}) = 0$.

Example: graph G_R In Figure 4.10, we display G_R, an example of a DAG learned on the variables $\mathbf{X}_{\mathcal{R}(F)} = (X_{11}, X_{21})$. Note that this graph is complete, and therefore maximizes the MLL score w.r.t. $D_{\mathcal{R}(F)}$ in this particular case.

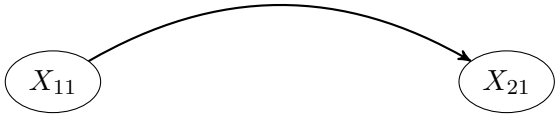


Figure 4.10: Graph G_R

Example: graph G^* as in Proposition 4 Figure 4.11 represents the graph $G^* = G_R \cup F$. Assuming `sota-BNSL` returns G_R displayed in Figure 4.10, the graph G^* is returned by the `ds-BNSL` algorithm.

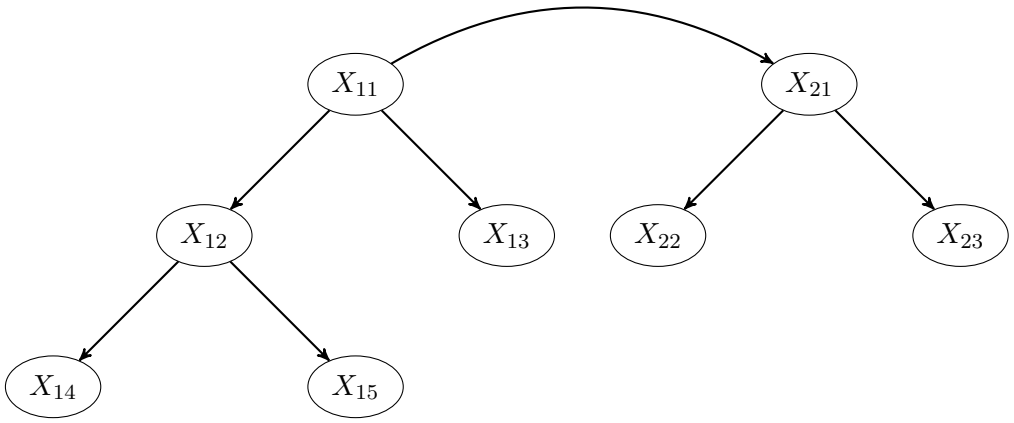


Figure 4.11: Graph G^*

Proposition 4, associated with the fact that G_R is a complete graph, shows that G^* maximizes the MLL score among all DAG structures encoding a distribution of $\mathbf{X}$.

Example: graph $\tilde{G}^*$ that still maximizes the MLL score Now suppose we have an additional relationship that holds in D:

$$H^D(X_{21}|X_{13}) = H^D(X_{21}|X_{11}). \tag{4.11}$$

This equation expresses the fact that the variable X_{13} is as useful as X_{11} in terms of information brought concerning variable X_{21}, despite encoding a smaller amount of information than X_{11} (Lemma 2.3 in Chapter 2 gives that $H^D X_{11} > H^D(X_{13})$). This case is reasonably common in practice on data that contain a lot of determinism (such as IoT datasets).
For example, X_{11} could represent a building room, whereas X_{13} represents the room orientation (N,S,W,E): most of the time, the nodes lower in the trees encode more general and 'pure' concepts than the roots.

We define the DAG $\tilde{G}^*$, displayed in Figure 4.12, as the graph G^* for which the arc (v_{11}, v_{21}) is removed and replaced by the arc (v_{11}, v_{21}), *i.e.* the arc (v_{11}, v_{21}) is lowered to (v_{11}, v_{21}).

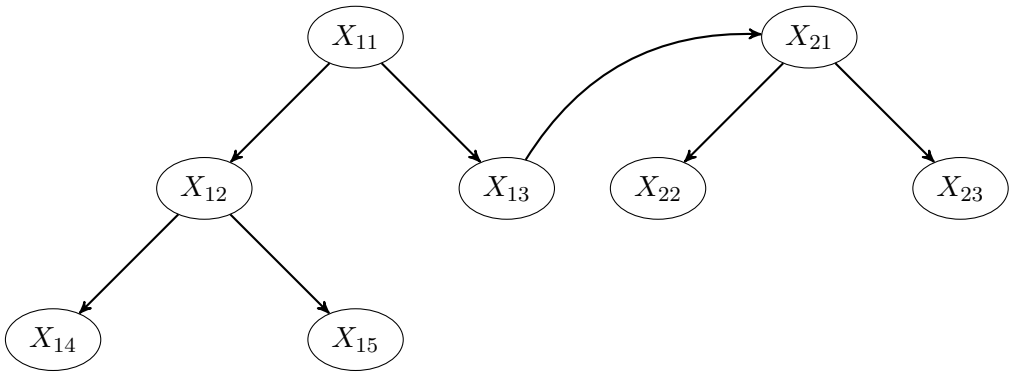


Figure 4.12: Graph $\tilde{G}^*$

A straightforward computation shows that:

$$\begin{aligned} s_D^{MLL}(\tilde{G}^*) &= s_D^{MLL}(G^*) - M\left(\underbrace{H^D(X_{21}|X_{13})}_{\text{added arc}} - \underbrace{H^D(X_{21}|X_{11})}_{\text{removed arc}}\right) \\ &= s_D^{MLL}(G^*). \end{aligned}$$

The **DAG $\tilde{G}^*$ therefore also maximizes the MLL score**.

Moreover, $\tilde{G}^*$ **has a smaller number of parameters than** G^*:

$$\begin{aligned} \mathcal{P}(\tilde{G}^*) - \mathcal{P}(G^*) &= |Val(X_{21})|(|Val(X_{11})| - 1) - |Val(X_{21})|(|Val(X_{13})| - 1) \\ &= |Val(X_{21})| \underbrace{(|Val(X_{11})| - |Val(X_{13})|)}_{>0 \text{ from Lemma 8}} \\ &> 0. \end{aligned}$$

Therefore, $\tilde{G}^*$ is strictly simpler than G^*, but has the same quantitative performance.

In the following section, we propose an algorithm that automatizes the arc lowering search idea. We then show how such an algorithm can be generalized to graphs formed from quasi-deterministic forests such as those returned by the `QuasiDeterScreen` algorithm.

2.1.3 Automation of this idea: the `LoweringArcs` algorithm

Similarly as in the case of a single deterministic tree, we can potentially find several graphs maximizing the MLL score by performing local search with the graph returned by the `ds-BNSL` algorithm as a starting point (as in the example).

Let us consider the general case, where D contains M observations of $\mathbf{X} = (X_1, \ldots, X_n)$. Let G^* be a graph that maximizes the MLL score, and $LocalMaxMLL(G^*)$ a set of DAGs that also maximize the MLL score and that can be found by searching for arcs to lower in the graph G^*. By definition of the 'lowering' operation,

$$LocalMaxMLL(G^*) \subset \underset{G \in DAG_{[\![1,n]\!]}}{\operatorname{argmax}} s_D^{MLL}(G).$$

In an analogous approach as the one presented in equation 2.5 in Chapter 2, we are interested in finding a solution $\tilde{G}^*$ with minimal lower of parameters in $LocalMaxMLL$, *i.e.*

$$\tilde{G}^* \in \underset{G \in LocalMaxMLL(G^*)}{\operatorname{argmin}} \mathcal{P}(G).$$

Note that this is equivalent in that case to:

$$\tilde{G}^* \in \underset{G \in LocalMaxMLL(G^*)}{\operatorname{argmax}} s_D^{BIC}(G). \tag{4.12}$$

This problem can be solved in a quadratic worst case complexity (with respect to the number of variables), as long as the local search is done one variable at a time.[8] For this purpose, we propose the `LoweringArcs` algorithm, presented in Algorithm 13, which takes for input:

- a deterministic forest $F = ([\![1, n]\!], A_F)$,
- a DAG structure $G_R = (\mathcal{R}(F), A_R)$,
- a dataset D containing observations of the modeled variables $\mathbf{X}$.

Algorithm 13 `LoweringArcs`

Input: $F = ([\![1, n]\!], A_F)$, $G_R = (\mathcal{R}(F), A_R)$, D
1: $G_R^{new} \leftarrow G_R$
2: **for** $r \in \mathcal{R}(F)$ **do**
3: **for** $\pi_r \in \pi^{G_R}(r)$ **do** *#for each root arc, try to lower it*
4: $\pi_{oth}(r) \leftarrow \pi^{G_R}(r) \setminus \pi_r$
5: $\pi_{better}(r) \leftarrow \{v \in \texttt{Desc}^F(\pi_r) | H^D(X_r | X_v, \mathbf{X}_{\pi_{oth}(r)}) = H^D(X_r | \mathbf{X}_{\pi^{G_R}(r)})\}$
6: **if** $\pi_{better}(r) \neq \emptyset$ **then**
7: $\pi_r^* \leftarrow \underset{v \in \pi_{better}(r)}{\operatorname{argmin}} |Val(X_v)|$
8: $G_R^{new} \leftarrow (\mathcal{R}(F), A_R \setminus \{(\pi_r, r)\} \cup \{(\pi_r^*, r)\})$
9: $G^{new} \leftarrow F \cup G_R^{new}$
Output: G^{new}

where $\texttt{Desc}^F$ on line 5 refers to the function that returns the set of descendants in F. In the case of a root node $\pi_r \in \mathcal{R}(F)$, the set $\texttt{Desc}^F(\pi_r)$ corresponds to the set of all the nodes in the corresponding deterministic tree (except for its root π_r).

[8] Meaning that, for roots that have several parents in G_R, we only try to lower the arcs leading to this root one by one, without considering possible interactions in between the parents. This explores a smaller local space around G^* than if we searched into parent sets, but proved in practice to be sufficient to detect interesting arc lowering possibilities, while being computationally efficient ($\mathcal{O}(n^2)$).

2.1.4 Generalization to the case of `qds-BNSL`

In the more general case of the `qds-BNSL` algorithm several points do not hold anymore, and notably:

- we do not have any score-maximizing guarantees for the structure returned by the `qds-BNSL` algorithm,
- we do not have the guarantee that a non-root variable is simpler (in terms of number of values) than its corresponding root variable.

Therefore, we cannot be sure to simplify the overall structure while not decreasing the MLL score simply by looking at empirical conditional entropies as we do in Algorithm 13. However, we can give ourselves a criterion that takes into account both fit and complexity of structures (such as the BIC score, as suggested by Equation (4.12)), and design a slightly modified algorithm, that uses this criterion for searching over potential arc lowering operations.

We propose the `LoweringArcsGeneral` algorithm, presented in Algorithm 14, which takes for input:

- a deterministic forest $F = (\llbracket 1, n \rrbracket, A_F)$,
- a DAG structure $G_R = (\mathcal{R}(F), A_R) \in DAG_{\mathcal{R}(F)}$,
- a dataset D containing observations of the modeled variables $\mathbf{X}$,
- a criterion $\mathtt{c}_D$, taking for input (in this order):
  - a node $v \in V$
  - a set of potential parents for v $\pi(v) \in 2^V$.

This algorithm has the same structure as Algorithm 13, except for the $\mathtt{c}_D$ input, which changes the potential better parent selection (lines $5-7$). In Algorithm 13, this is done first by identifying variables with the same conditional entropy, then by choosing the simplest, while it is now performed by directly evaluating the criterion $\mathtt{c}_D$ for every descendant of the considered root variable.
Note that using the criterion defined as:

$$\mathtt{c}_D(v, \pi(v)) = s^{BIC}_{D_{v \cup \pi(v)}}((v \cup \pi(v), \{(u, v) | u \in \pi(v)\}))$$

in Algorithm 14 is equivalent to Algorithm 13 (assuming the input forest F is deterministic), as stated in Equation (4.12).

Algorithm 14 `LoweringArcsGeneral`

Input: $F = (\llbracket 1, n \rrbracket, A_F)$, $G_R = (\mathcal{R}(F), A_R)$, D, $\mathtt{c}_D$

1: $G_R^{new} \leftarrow G_R$
2: **for** $r \in \mathcal{R}(F)$ **do**
3: **for** $\pi_r \in \pi^{G_R}(r)$ **do**
4: $\pi_{oth}(r) \leftarrow \pi^{G_R}(r) \setminus \pi_r$
5: $\pi_{better}(r) \leftarrow \{v \in \mathtt{Desc}^F(\pi_r) | \mathtt{c}_D(r, \pi_{oth}(r) \cup \{v\}) > \mathtt{c}_D(r, \pi^{G_R}(r))\}$
6: **if** $\pi_{better}(r) \neq \emptyset$ **then**
7: $\pi_r^* \leftarrow \underset{v \in \pi_{better}(r)}{\text{argmax}}\ \mathtt{c}_D(r, \pi_{oth}(r) \cup \{v\})$
8: $G_R^{new} \leftarrow (\mathcal{R}(F), A_R \setminus \{(\pi_r, r)\} \cup \{(\pi_r^*, r)\})$
9: $G^{new} \leftarrow F \cup G_R^{new}$

Output: G^{new}

2.1.5 Complexity considerations

The number of operations made in Algorithms 13-14 is given by:

$$\begin{aligned} C(\mathtt{LoweringArcsGeneral}) &\leq \underbrace{\sum_{r \in \mathcal{R}(F)}}_{\text{line 2}} \underbrace{\sum_{r' \in \mathcal{R}(F)}}_{\text{line 3}} \underbrace{|\mathtt{Desc}^F(r')|}_{\text{line 5}} \\ &\leq \underbrace{|\mathcal{R}(F)|}_{\leq n} \times n. \end{aligned}$$

Which proves that these algorithms complexity is $\mathcal{O}(n^2)$, which is consistent with the screening algorithms `DeterScreen` and `QuasiDeterScreen` respectively presented as Algorithm 5 and Algorithm 7 in Chapter 2.

2.2 Choosing ε for the `qds-BNSL` algorithm

2.2.1 Ad-hoc approach and limitations

In the experiments presented in Section 5 of Chapter 2, we recall that ε used in the `qds-BNSL` algorithm was chosen with an ad-hoc approach: we pick values for $n_r(\varepsilon)$, the number of variables we want to set aside with `QuasiDeterScreen`, and we manually find the corresponding values for ε.

Formally, for a given dataset D and $x \in [0, 1]$, we define $\varepsilon_x = n_r^{-1}(\lfloor xn \rfloor)$: ε_x is the value of ε for which the number of roots of the quasi deterministic forest F_ε represents a proportion x of the total number of variables.

The computation of ε_x is not problematic: once $\mathbb{H}^D$ is computed and stored, evaluating $n_r(\varepsilon)$ is done in constant time, and finding one of $n_r(\cdot)$'s quantiles is doable in $\mathcal{O}(\log(n))$ operations (dichotomy), which is negligible compared to the overall complexity of the screening phase ($\mathcal{O}(n^2)$).

However this approach has limitations: as the results presented in Section 5 of Chapter 2 show, the impact of removing a given proportion of the variable using screening depends highly

on the dataset. As shown in Table 2.4, for some datasets, the final graph performance is almost not impacted (less than 2%) when setting aside half of the variables (munin3, pump it up), whereas for some other datasets, we have up to $10-15\%$ percent of BDe and CVLL score loss (covertype, andes, plants).

In this subsection, we propose a way to **graphically anticipate the potential of the quasi-determinism screening approach** for a given dataset, and to *choose* ε in cases where `qds-BNSL` is promising. For this purpose, we only use values that can be computed in constant time from the empirical conditional entropy matrix

$$\mathbb{H}^D = (H^D(X_i|X_j))_{1\leq i,j\leq n}.$$

2.2.2 Graphical estimation of algorithm's potential

We consider a dataset D containing M observations of variables $\mathbf{X} = (X_1, \ldots, X_n)$, and an associated empirical conditional entropy matrix $\mathbb{H}^D = (H^D(X_i|X_j))_{1\leq i,j\leq n}$.
In Section 6 of Chapter 2, we suggest that `qds-BNSL` appears in its best light in a scenario for which the quasi-determinism screening phase enables the elimination of an important proportion of the variables (to secure an important time gain), for ε reasonably *small* (to guarantee a controlled performance loss).
The natural question that comes to mind is: what does **small** mean with respect to the value of ε ? Intuitively, we want ε to be small compared to the values of the variables entropies.
Indeed, for a **given** $i \in [\![1, n]\!]$, the information given by the fact that

$$\exists j \in [\![1, n]\!] \setminus \{i\} \text{ such that } H^D(X_i|X_j) \leq \varepsilon \tag{4.13}$$

is all the more interesting that ε is small compared to $H^D(X_i)$.

Indeed, if $\varepsilon \approx H^D(X_i)$, Equation (4.13) holds for any variable X_j which is slightly correlated to X_i. On the contrary, if $\varepsilon << H^D(X_i)$, Equation (4.13) implies that X_j brings a lot of information with regards to the value of the variable X_i.
For this reason, several approaches such as the one from Chow and Liu (1968) or the more recent one from Cheng et al. (1997) consider the empirical mutual information, *i.e.* $I^D(X_i, X_j) = H^D(X_i) - H^D(X_i|X_j)$ as a criterion for structure learning. However,

- this quantity is symmetric in (X_i, X_j), which is not sought-after in the context of our screening algorithm, since we wish to find 'representative' variables (roots of the forest) that contain most of the information associated with a subgroups of other variables,
- results that involve empirical conditional entropy, such as Proposition 3, Proposition 4 or Proposition 8 cannot be simply extended, to our knowledge, to mutual information.

We therefore choose to compare, for a given value ε, the proportion of the variables that are not set aside by the `QuasiDeterScreen` algorithm (*i.e.* variables corresponding to the roots of $F_\varepsilon = \texttt{QuasiDeterScreen}(D, \varepsilon)$) to the proportion of variables which have an entropy bigger than ε.
Formally, we display the simultaneous evolution of the two following dataset-dependent quantities with respect to ε, that both have values in $[0, 1]$:

- $\texttt{RemainingVariables}^D$, is the proportion of root variables left by `QuasiDeterScreen`. For any $\varepsilon \geq 0$,
$$\texttt{RemainingVariables}^D(\varepsilon) = \frac{n_r(\varepsilon)}{n} = \frac{|\mathcal{R}(F_\varepsilon)|}{n}.$$
- $\texttt{EntropyRepartition}^D$ is the proportion of variables with entropy bigger than ε. For any $\varepsilon \geq 0$,
$$\texttt{EntropyRepartition}^D(\varepsilon) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{H^D(X_i)\geq\varepsilon\}}.$$

For a given dataset D, we expect these two quantities to be all the more far apart (although they both go towards 0 when ε grows) that the impact of running the `qds-BNSL` algorithm on D is promising. Intuitively, an interesting choice for ε is a value for which $\texttt{EntropyRepartition}^D(\varepsilon)$ is the biggest possible, and $\texttt{RemainingVariables}^D(\varepsilon)$ is the smallest possible.

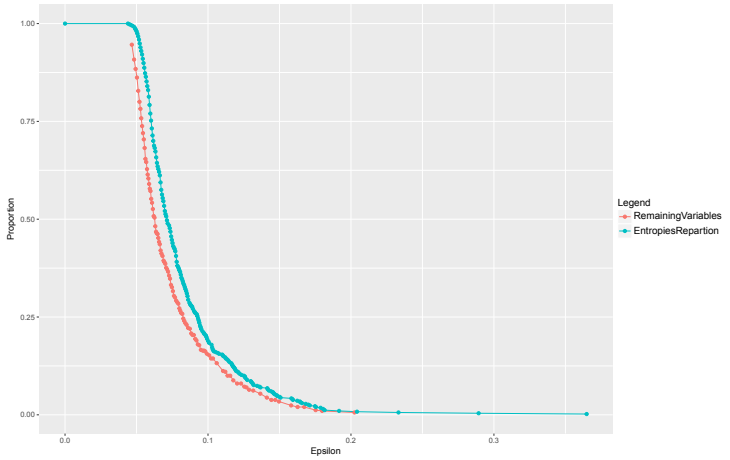
A rule of thumb for the choice of ε is to pick a value ε_0 for which the difference

$$\texttt{EntropyRepartition}^D(\varepsilon_0) - \texttt{RemainingVariables}^D(\varepsilon_0)$$

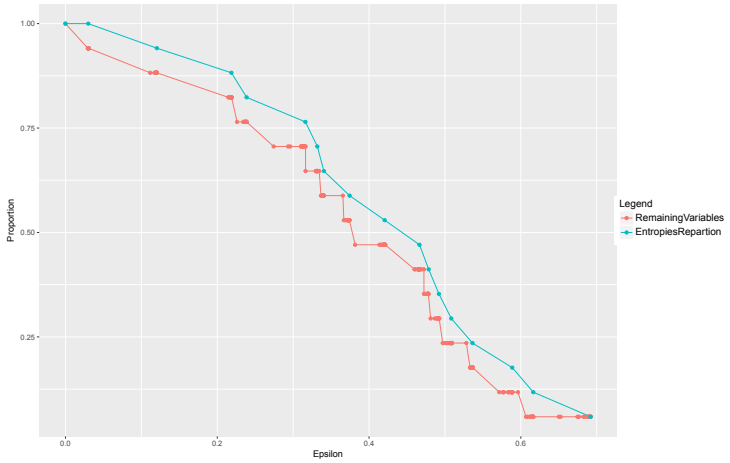
is the most important: for $\varepsilon = \varepsilon_0$ we eliminate a lot of variables (and therefore win a lot of time), but the variables we eliminate are promising in terms of how well they are explained by theirs parents in F_{ε_0} (and we are therefore confident on how well this forest will perform overall). The best case is reached when a lot of variables are eliminated for a very small epsilon, *i.e.* when there are a lot of real pairwise determinism in the data (which is the case for piu and even more for the dataset extracted from `HOMES` metadata).

Figure 4.13 displays such plots for 3 datasets out of those that are studied in Section 5 of Chapter 2, and on which the `qds-BNSL` algorithm has different levels of performance:

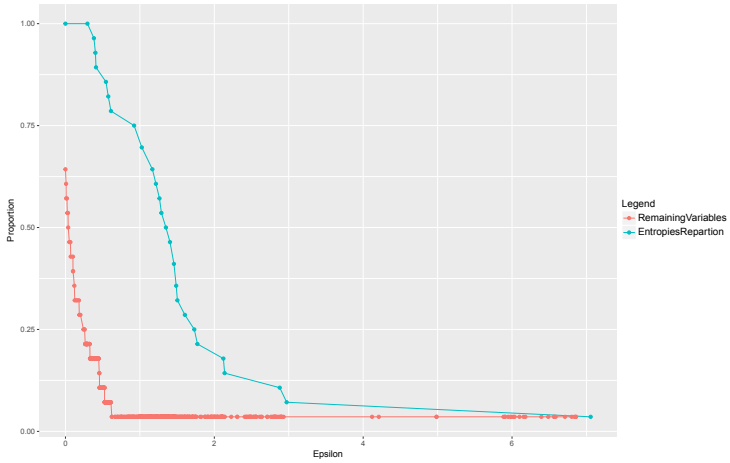
- *book* dataset (Figure 4.13(a)), for which the algorithm `qds-BNSL` seems to be moderately adapted from the results displayed in Section 5 of Chapter 2,
- *msnbc* (Figure 4.13(b)), for which the algorithm `qds-BNSL` seems to be well adapted,
- *pumpitup* (Figure 4.13(c)), for which the algorithm `qds-BNSL` seems to be very well adapted.



(a) Candidate criterion for choice of ε - *book* dataset



(b) Candidate criterion for choice of ε - *msnbc* dataset



(c) Candidate criterion for choice of ε - *pumpitup* dataset

Figure 4.13: Candidate graphical criterion for the choice of ε on three selected datasets

This intuition is confirmed by the three displayed graphs in Figure 4.13: the more these graphs have potential with regards to the `qds-BNSL` algorithm, the more the two quantities are far apart.
In the case of the pump it up dataset, we see that $\varepsilon_{0.5}$ is very small compared to the values of the entropies ($\varepsilon_{0.5} \approx 0.05$), which suggests that we could have chosen ε bigger than $\varepsilon_{0.5}$ and possibly obtain even better results.

Although the link between the quantity $\texttt{EntropyRepartition}^D(\varepsilon) - \texttt{RemainingVariables}^D(\varepsilon)$ and the efficiency of the `qds-BNSL` algorithm should be studied more in depth (for example by building a more complex criterion), this appears as an interesting first approach (and has quadratic complexity).

3 (Quasi-)determinism screening and the BIC score: prospective results

For now, although we have made an emphasis on minimizing the number of parameters of the learned models, the results we have proven mostly concern the MLL score. We are now seeking to prove similar results for scores that naturaly imply a fit/complexity tradeoff. In Section 1.4 we mention possible extensions of results to the VLL score. In this section we focus on the BIC score, that has an expression very close to that of the MLL score: we present some ideas of potential results, and pending questions.

3.1 BIC score: generalization

We suppose we have a dataset D containing M observation of $\mathbf{X} = (X_1, \ldots, X_n)$, and that $V = [\![1, n]\!]$. We remind that the BIC score for Bayesian networks is expressed as a penalization of the MLL score relatively to the number of parameters of the model:

$$\forall G \in DAG_V,\ s_D^{BIC}(G) = s_{DAG_V}^{MLL}(G) - \frac{\log(M)}{2}\mathcal{P}(G).$$

where $\mathcal{P}(G)$ is the number of parameters of the Bayesian network with structure G.

For any $\lambda \geq 0$ we define the $BIC(\lambda)$ score that generalizes the BIC score as follows:

$$\forall G \in DAG_V,\ s_D^{BIC(\lambda)}(G) = s_D^{MLL}(G) - \lambda\frac{\log(M)}{2}\mathcal{P}(G).$$

In this context, λ acts as a regularization hyperparameter. We notably have:

- When $\lambda \longrightarrow 0$, then for any $G \in DAG_V$, $s_D^{BIC(\lambda)}(G) \longrightarrow s_D^{MLL}(G)$, and the $BIC(\lambda)$ score becomes equivalent to the MLL score which leads to complete DAGs as mentioned previously.

- For $\lambda = 1$, the $BIC(\lambda)$ score is the usual BIC score for Bayesian network structures.
- For $\lambda = \frac{2}{\log(M)}$, the $BIC(\lambda)$ score is the AIC score for Bayesian network structures.
- When $\lambda \longrightarrow +\infty$, the $BIC(\lambda)$ score increasingly favors empty DAG structures.

3.2 Determinism and generalized BIC score: open questions

For $\lambda > 0$, maximizing $s^{BIC(\lambda)}$ among all structures $G \in DAG_V$ is a NP-Hard problem. When in presence of deterministic relations in the data, we would like to be able to use them to speed up the structure learning task with provably no impact on the overall score, like we proved possible when using the MLL score as a target (*i.e.* $BIC(\lambda = 0)$).

We have identified some open questions that seem interesting in going forward down the path leading to some guarantees concerning (quasi-)determinism screening in the context of the BIC score. As with the MLL score, we are first interested in what happens in the particular case where a deterministic tree exists, hoping that we will then be able to generalize to deterministic forests.

Suppose that a deterministic tree T exists w.r.t. D. Proposition 3 states that T maximizes the $BIC(\lambda = 0)$ score: the MLL score. Since the number of structures is finite, the set

$$S_D = \{\lambda^* > 0 \mid \forall \lambda \leq \lambda^*,\ T \in \underset{G \in DAG_V}{\operatorname{argmax}}\ s_D^{BIC(\lambda)}(G)\}$$

is nonempty (and obviously upper-bounded).
Therefore, we may define $\lambda^D_{max} = \max S_D$. By definition, $\lambda^D_{max} > 0$ and

$$\forall \lambda \leq \lambda^D_{max},\ T \in \underset{G \in DAG_V}{\operatorname{argmax}}\ s_D^{BIC(\lambda)}(G). \tag{4.14}$$

Open questions

- Can λ^D_{max} be found in reasonable computation time (without running a structure learning algorithm) ?
- Can λ^D_{max} be lower-bounded thanks to quantities such as $H^D(X_i)$ and/or $H^D(X_i|X_j)$ for $i, j \in V$?

Answering these questions is very interesting for two main reasons:

- Intuitively, deterministic trees are interesting because they maximize the MLL score (quantitative performance) while still satisfying sparsity assumptions (qualitative performance, or interpretability). The value of λ^D_{max} would enable to quantify the qualitative performance of T: the higher λ^D_{max}, the best the tradeoff realized by T.
  Notably, if $\lambda^D_{max} \geq 1$, then T maximizes the BIC (*i.e.* $BIC(\lambda = 1)$) score on DAG_V !

- Estimating λ^D_{max} would make it easier to find a more general result in the case of deterministic forests: if all the trees of a deterministic forest F maximize not only the MLL score but also the $BIC(\lambda^D_{max})$ score, it would be legitimate to use the $BIC(\lambda^D_{max})$ score as a target for running `sota-BNSL` on the root variables in order to find G_R[9], which we are confident could enable to have guarantees relative to a version BIC score for the graph $G_R \cup F$.

[9]Indeed, the `sota-BNSL` algorithms we use in practice inside `ds-BNSL` and `qds-BNSL` do not have the MLL as a target score as would suggest Proposition 4, since we know this leads to a complete DAG. Proving such results would enable us to know which target score to legitimately use for `sota-BNSL` that is run on the root variables.

Conclusion

Summary of contributions

This thesis addresses the goal of static and temporal data fusion in the context of IoT data. Such data seems to constitute an always increasing proportion of the available data today.[10].
Using the Bayesian network formalism described in Chapter 1, we presented several contributions building towards the goal of static and temporal data fusion.

In Chapter 2, we first explained how static data available from IoT systems may generate issues in the context of Bayesian network structure learning, principally caused by the presence of redundancy and determinism. We then proposed the `ds-BNSL` Bayesian network structure learning algorithm solving these issues, and extended it to a more general `qds-BNSL` algorithm that is not restricted to data containing strict determinism.

In Chapter 3, we then presented hybrid static-dynamic Bayesian networks (HSDBN), that jointly models static and temporal data. We proposed an algorithm to learn such networks from static metadata and temporal data that are available in practice from IoT systems, leveraging the `(q)ds-BNSL` algorithms introduced in Chapter 2. We also designed an inference algorithm tackling the problem of metadata recovery from a sequence of observations of temporal data alone.

Finally, in Chapter 4, we discussed the pertinence of the VLL score to evaluate Bayesian networks, and proposed some short-term algorithmic and theoretical perspectives, as well as leads for their further exploration.

Perspectives

Short term perspectives

Next steps primarily involve (i) obtaining results for the algorithms proposed in Chapter 3 on very large real-world IoT datasets, enabling to show the potential of HSDBNs in terms of applicability, and (ii) optimizing our code. For instance, parallelizing and programming the computationally

[10] `https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8\-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016`

intensive routines in a lower programming language (such as `C`), should give an even better edge to our proposed algorithms relatively to the already optimized `sota-BNSL` baseline we compare to.

Several perspectives mentioned in Chapter 4 also appear reasonably accessible. We are confident that results concerning guarantees of the `ds-BNSL` algorithm with respect to the VLL score or the (generalized) BIC score can be obtained (under some assumptions on the data), and we plan to run experiments on selected examples to get better insight on this question. Moreover, we are currently developing a algorithm based on the lowering arcs idea presented in Section 2.1.

Longer term perspectives

As mentioned in Section 6 of Chapter 2, we also have some perspectives concerning the `qds-BNSL` algorithm, such as extending the screening phase to relationships that involve more than two variables, or exploiting an alternative definition of determinism (for example using a normalized entropy criterion), as well as finding a principle way to set the ε hyperparameter.

We also believe that interesting theoretical links may be drawn to mutual-information based approaches such as the one from Chow and Liu (1968). Notably, it appears clearly that under the assumption of existence of a deterministic tree made in Proposition 3 in Chapter 2, the tree search algorithm by Chow and Liu returns one of the existing deterministic trees, as does `ds-BNSL`. Studying the theoretical similarities in the more general case of a (quasi-)deterministic forest is a promising perspective.

Moreover, we are looking forward to extending our algorithms and models to more complex (higher dimensional) data. For example, a HSDBN could be used to model a video, by using state of the art methods to extract high level features from successive frames (Sharif Razavian et al., 2014; Zeiler and Fergus, 2014), and considering these features as temporal variables.

Lastly, we could extend HSDBNs to handle multiple time scales, getting inspiration from works on multi-scale dynamic Bayesian networks (Chen and Wang, 2010). This could be particularly useful to model several scales of seasonality (day, week, year), as well as capturing in a less binary way the full spectrum of data temporality, from immutable to highly volatile.

Bibliography

Susanne G Bøttcher, Claus Dethlefsen, et al. *deal: A package for learning Bayesian networks*. Department of Mathematical Sciences, Aalborg University, 2003.

RR Bouckaert. *Bayesian belief networks: from inference to construction*. PhD thesis, Faculteit Wiskunde en Informatica, Utrecht University, 1995.

Wray Buntine. Theory refinement on Bayesian networks. In *Uncertainty Proceedings 1991*, pages 52–60. Elsevier, 1991.

Gilles Celeux and Jean-Baptiste Durand. Selecting hidden markov model state number with cross-validated likelihood. *Computational Statistics*, 23(4):541–564, 2008.

Feng Chen and Wei Wang. Activity recognition through multi-scale dynamic bayesian network. In *Virtual Systems and Multimedia (VSMM), 2010 16th International Conference on*, pages 34–41. IEEE, 2010.

Xue-Wen Chen, Gopalakrishna Anantha, and Xiaotong Lin. Improving Bayesian network structure learning with mutual information-based node ordering in the K2 algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):628–640, 2008.

Jie Cheng, David A Bell, and Weiru Liu. Learning belief networks from data: An information theory based approach. In *Proceedings of the sixth international conference on Information and knowledge management*, pages 325–331. ACM, 1997.

David Maxwell Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence, Montreal, QU*, pages 87–98, August 1995.

David Maxwell Chickering. Learning Bayesian networks is NP-complete. *Learning from data: Artificial intelligence and statistics V*, 112:121–130, 1996.

C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

Barry R Cobb, Rafael Rumí, and Antonio Salmerón. Bayesian network models with discrete and continuous variables. In *Advances in probabilistic graphical models*, pages 81–102. Springer, 2007.

Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, 42(2-3):393–405, March 1990. ISSN 0004-3702.

Gregory F Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

Robert G. Cowell. Conditions under which conditional independence and scoring methods lead to identical selection of Bayesian network models. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 91–97, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1.

James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 153–160, Arlington, Virginia, United States, 2011. AUAI Press. ISBN 978-0-9749039-7-2.

Jesse Davis and Pedro Domingos. Bottom-up learning of Markov network structure. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 271–278, 2010.

Cassio P de Campos, Mauro Scanagatta, Giorgio Corani, and Marco Zaffalon. Entropy-based pruning for learning Bayesian networks using bic. *Artificial Intelligence*, 260:42–50, 2018.

Cassio P de de Campos and Qiang Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12(Mar):663–689, 2011.

Luis M de Campos. A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7(Oct):2149–2187, 2006.

Sergio Rodrigues de Morais, Alexandre Aussem, and Marilys Corbex. Handling almost-deterministic relationships in constraint-based Bayesian network discovery: Application to cancer risk factor identification. In *European Symposium on Artificial Neural Networks, ESANN'08*, 2008.

Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Comput. Intell.*, 5(3):142–150, December 1989. ISSN 0824-7935. doi: 10.1111/j.1467-8640.1989.tb00324.x.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

Chloé Desdouits, Jean-Louis Bergerand, Pierre-Alexis Berseneff, Claude Le Pape, and Dimitri Yanculovici. Energy study of a manufacturing plant. *ECEEE Industrial Efficiency Summer Study. Berlin, Germany*, 2016.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Martin Diaz, Gonzalo Juan, Oikawa Lucas, and Alberto Ryuga. Big data on the internet of things: An example for the e-health. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 898–900. IEEE, 2012.

Charbel El Kaed, Brett Leida, and Tony Gray. Building management insights driven by a multi-system semantic representation approach. In *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*, pages 520–525. IEEE, 2016.

Nicholas Etherden, Anders Kim Johansson, Ulf Ysberg, Kjetil Kvamme, David Pampliega, and Craig Dryden. Enhanced lv supervision by combining data from meters, secondary substation measurements and medium voltage supervisory control and data acquisition. *CIRED-Open Access Proceedings Journal*, 2017(1):1089–1093, 2017.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

Nir Friedman, Iftach Nachman, and Dana Peér. Learning Bayesian network structure from massive datasets: the 'sparse candidate 'algorithm. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 206–215. Morgan Kaufmann Publishers Inc., 1999.

Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

Robert Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Machine Intelligence and Pattern Recognition*, volume 10, pages 209–219. Elsevier, 1990.

David Heckerman. A tutorial on learning with Bayesian networks. In *Learning in graphical models*, pages 301–354. Springer, 1998.

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

Max Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Machine Intelligence and Pattern Recognition*, volume 5, pages 149–163. Elsevier, 1988.

Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

JinHyung Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *International Joint Conference on Artificial Intelligence*, pages 0–0, 1983.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Dan D Koo, John J Lee, Aleksei Sebastiani, and Jonghoon Kim. An internet-of-things (iot) system development and implementation for bathroom safety enhancement. *Procedia Engineering*, 145:396–403, 2016.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Vincenzo La Tosa, Sylvain Maré, Franck Bernier, and Daniel Piette. Pervasive energy measurements for buildings monitoring. In *Proceedings of the second Workshop on eeBuildings Data Models*, 2011.

Wei Luo. Learning Bayesian networks in semi-deterministic systems. In *Canadian Conference on AI*, pages 230–241. Springer, 2006.

Ahmed Mabrouk, Christophe Gonzales, Karine Jabet-Chevalier, and Eric Chojnacki. An efficient Bayesian network structure learning algorithm in the presence of deterministic relations. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 567–572. IOS Press, 2014.

Nikos Mamoulis, Ling Liu, and M. Tamer Özsu. *Temporal Data Mining*, pages 2948–2952. Springer US, Boston, MA, 2009.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Kevin Patrick Murphy and Stuart Russell. Dynamic bayesian networks: representation, inference and learning. 2002.

K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning. MIT Press, 2012.

Radhakrishnan Nagarajan, Marco Scutari, and Sophie Lèbre. Bayesian networks in r. *Springer*, 122:125–127, 2013.

Hala Najmeddine, Frédéric Suard, Arnaud Jay, Philippe Marechal, and Marié Sylvain. Mesures de similarité pour l'aide à l'analyse des données énergétiques de bâtiments. In *RFIA 2012 (Reconnaissance des Formes et Intelligence Artificielle)*, pages 978–2, 2012.

Le T Nguyen, Pang Wu, William Chan, Wei Peng, and Ying Zhang. Predicting collective sentiment dynamics from time-series social media. In *Proceedings of the first international workshop on issues of sentiment discovery and opinion mining*, page 6. ACM, 2012.

Siqi Nie, Cassio Polpo de Campos, and Qiang Ji. Learning Bayesian networks with bounded tree-width via guided search. In *AAAI*, pages 3294–3300, 2016.

Dávid Pál, Barnabás Póczos, and Csaba Szepesvári. Estimation of rényi entropy and mutual information based on generalized nearest-neighbor graphs. In *Advances in Neural Information Processing Systems*, pages 1849–1857, 2010.

Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10): 1082–1093, 2015.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7.

Judea Pearl et al. Causal inference in statistics: An overview. *Statistics surveys*, 3:96–146, 2009.

Peter Pflaum, Mazen Alamir, and Mohamed Yacine Lamoudi. Probabilistic energy management strategy for ev charging stations using randomized algorithms. *IEEE Transactions on Control Systems Technology*, 2017.

Kimball Ralph. The data warehouse toolkit: Practical techniques for building dimensional data warehouses, 1996.

Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Advances in neural information processing systems*, pages 294–300, 2001.

Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

Dmitry Rusakov and Dan Geiger. Asymptotic model selection for naive Bayesian networks. *Journal of Machine Learning Research*, 6(Jan):1–35, 2005.

Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76, 1996.

Mauro Scanagatta, Cassio P de Campos, Giorgio Corani, and Marco Zaffalon. Learning Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems*, pages 1864–1872, 2015.

Mauro Scanagatta, Giorgio Corani, Cassio P de Campos, and Marco Zaffalon. Learning treewidth-bounded Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems*, pages 1462–1470, 2016.

Richard Scheines, Peter Spirtes, Clark Glymour, Christopher Meek, and Thomas Richardson. Tetrad 3: Tools for causal modeling–users manual. *CMU Philosophy*, 1996.

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

Marco Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010. doi: 10.18637/jss.v035.i03.

Ross D Shachter and Mark A Peot. Simulation approaches to general probabilistic inference on belief networks. In *Machine Intelligence and Pattern Recognition*, volume 10, pages 221–231. Elsevier, 1990.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

Tomi Silander and Petri Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, UAI'06, pages 445–452. AUAI Press, 2006. ISBN 0-9749039-2-2.

Tomi Silander, Petri Kontkanen, and Petri Myllymäki. On sensitivity of the map Bayesian network structure to the equivalent sample size parameter. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 360–367. AUAI Press, 2007.

David J Spiegelhalter and Steffen L Lauritzen. Sequential updating of conditional probabilities on directed graphical structures. *Networks*, 20(5):579–605, 1990.

Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

Marc Teyssier and Daphne Koller. Ordering-based search: a simple and effective algorithm for learning Bayesian networks. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 584–590. AUAI Press, 2005.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

Ioannis Tsamardinos, Laura E Brown, and Constantin F Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

Jimmy Vandel, Brigitte Mangin, and Simon De Givry. New local move operators for Bayesian network structure learning. *Proceedings of PGM-12, Granada, Spain*, 2012.

Thomas Verma and Judea Pearl. Causal networks: semantics and expressiveness. In *UAI*, pages 69–78. North-Holland, 1988.

Sandeep Yaramakala and Dimitris Margaritis. Speculative Markov blanket discovery for optimal feature selection. In *Fifth IEEE international conference on Data Mining*, pages 4–pp. IEEE, 2005.

Changhe Yuan, Brandon Malone, et al. Learning optimal Bayesian networks: A shortest path perspective. *Journal of Artificial Intelligence Research*, 2013.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Nevin L Zhang and David Poole. A simple approach to Bayesian network computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 1994.

Proofs

1 Proofs of results presented in Chapter 1

Proof of Lemma 1 The log function being concave, $-\log$ is convex. For any $p, q \in \mathcal{F}_P(U)$, Jensen's inequality gives:

$$\begin{aligned}\sum_{u\in U} p(u)(-\log)\left(\frac{q(u)}{p(u)}\right) &\geq (-\log)\left(\sum_{u\in U} p(u)\frac{q(u)}{p(u)}\right)\\ &= -\log\left(\sum_{u\in U} q(u)\right)\\ &= 0.\end{aligned}$$

Therefore $\forall p, q \in \mathcal{F}_P(U)$

$$\sum_{u\in U} p(u)\log\left(\frac{q(u)}{p(u)}\right) \leq 0$$

which directly gives:

$$\sum_{u\in U} p(u)\log(q(u)) \leq \sum_{u\in U} p(u)\log(p(u)).$$

■

Proof of Proposition 1 By definition, **for any parameter** Θ and thanks to the fact that the observations in D are considered *i.i.d*, the likelihood is expressed as:

$$\mathcal{L}_D(\Theta) = \prod_{m=1}^{M} P_\Theta(X = x^{(m)})$$

To simplify the maximization problem, we consider the log-likelihood $l_D(\Theta)$, that can be rewritten as follows:

$$\begin{aligned}
l_D(\Theta) &= \log(\mathcal{L}_D(\Theta)) \\
&= \log\left(\prod_{m=1}^{M} P_\Theta(X = x^{(m)})\right) \\
&= \sum_{m=1}^{M} \log(p_\Theta(x^{(m)})) \\
&= \sum_{m=1}^{M} \sum_{x \in Val(X)} \log(\underbrace{P_\Theta(X = x)}_{\theta_x}) \mathbb{I}_{x^{(m)}=x} \\
&= \sum_{x \in Val(X)} \sum_{m=1}^{M} \log(\theta_x) \mathbb{I}_{x^{(m)}=x} \\
&= \sum_{x \in Val(X)} \log(\theta_x) \underbrace{\left(\sum_{m=1}^{M} \mathbb{I}_{x^{(m)}=x}\right)}_{=C^D(x)} \\
&= \sum_{x \in Val(X)} C^D(x) \log(\theta_x).
\end{aligned}$$

For all $x \in Val(X)$, we define $\tilde{\Theta} = \{\tilde{\theta}_x\}_x$ where for all $x \in Val(X)$,

$$\tilde{\theta}_x = \frac{C^D(x)}{M}.$$

We immediately check that $\tilde{\Theta} \in \mathcal{F}_P(Val(X))$:

$$\begin{aligned}
\sum_{x \in Val(X)} \tilde{\theta}_x &= \frac{1}{M} \sum_{x \in Val(X)} \sum_{m=1}^{M} \mathbb{I}_{x^{(m)}=x} \\
&= \frac{1}{M} \sum_{m=1}^{M} \underbrace{\sum_{x \in Val(X)} \mathbb{I}_{x^{(m)}=x}}_{=1} \\
&= 1.
\end{aligned}$$

Using Lemma 1, and since θ and $\tilde{\theta}$ are in $\mathcal{F}_P(Val(X))$, for all $\Theta \in \mathcal{F}_P(Val(X))$ we have:

$$\begin{aligned}
l_D(\Theta) &= M \sum_{x \in Val(X)} \tilde{\theta}_x \log(\theta_x) \\
&\leq M \sum_{x \in Val(X)} \tilde{\theta}_x \log(\tilde{\theta}_x) \\
&= l_D(\tilde{\Theta}).
\end{aligned}$$

Thus, by definition,

$$\tilde{\Theta} \in \underset{\Theta \in \mathcal{F}^P(Val(X))}{\operatorname{argmax}} \; l_D(\Theta)$$

Since the log-likelihood is concave, it admits a unique maximum: the MLE $\hat{\Theta}^D$. This finally gives:

$$\hat{\Theta}^D = \left\{ \left(\frac{C^D(x)}{M} \right) \right\}_{x \in Val(X)}$$

■

2 Proofs of results presented in Chapter 2

Proof of Lemma 2 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 $\theta \in \Theta_G$,

$$\begin{aligned} l_D(\Theta) &= \sum_{m=1}^{M} \log(\underbrace{p_\theta(x_1^{(m)} \dots, x_n^{(m)})}_{\prod_{i=1}^n \theta_{x_i^{(m)}|\mathbf{x}_{\pi(i)}^{(m)}}}) \\ &= \sum_{m=1}^{M} \sum_{i=1}^{n} \log(\theta_{x_i^{(m)}|\mathbf{x}_{\pi(i)}^{(m)}}) \\ &= \sum_{i=1}^{n} \sum_{x_i, \mathbf{x}_{\pi(i)}} C^D(x_i, \mathbf{x}_{\pi(i)}) \log(\theta_{x_i|\mathbf{x}_{\pi(i)}}) \end{aligned}$$

where $C^D(\cdot)$ is the count function associated with D:
$\forall I \subset V$,

$$C^D(\mathbf{x}_I) = \sum_{m=1}^{M} \mathbb{I}_{\mathbf{x}_I^{(m)} = \mathbf{x}_I} = M p^D(\mathbf{x}_I).$$

Moreover, the maximum likelihood estimator θ for categorical variables is given by the local empirical frequencies (extension of Proposition 1) *i.e.*

$$\theta_{x_i|\mathbf{x}_{\pi(i)}} = p^D(x_i|\mathbf{x}_{\pi(i)}) = \frac{C^D(x_i, \mathbf{x}_{\pi(i)})}{C^D(\mathbf{x}_{\pi(i)})}.$$

Therefore we get:

$$\begin{aligned} s_D^{MLL}(G) &= \max_{\theta \in \Theta_G} l_D(\Theta) \\ &= l_D(\Theta) \\ &= \sum_{i=1}^{n} \sum_{x_i, \mathbf{x}_{\pi(i)}} C^D(x_i, \mathbf{x}_{\pi(i)}) \log(\theta_{x_i|\mathbf{x}_{\pi(i)}}) \\ &= \sum_{i=1}^{n} \sum_{x_i, \mathbf{x}_{\pi(i)}} M p^D(x_i, \mathbf{x}_{\pi(i)}) \log(p^D(x_i|\mathbf{x}_{\pi(i)})) \\ &= -M \sum_{i=1}^{n} H^D(X_i|\mathbf{X}_{\pi(i)}). \end{aligned}$$

■

Proof of Proposition 3 Let T be as in the hypothesis of Proposition 3, we can prove that the bound for the MLL score among DAGs proven in Lemma 5 is reached for T, which gives the result.

Without any loss of generality, let us suppose that T's root is 1. Then,

$$\begin{aligned} s_D^{MLL}(T) &= -M \sum_{i=1}^{n} H^D(X_i|\mathbf{X}_{\pi(i)}) \\ &= -M \left(H^D(X_1) + \sum_{i=2}^{n} \underbrace{H^D(X_i|\mathbf{X}_{\pi(i)})}_{=0} \right) \\ &\geq -M \; H^D(X_1, \ldots, X_n) \\ &= \max_{G \in DAG_V} s_D^{MLL}(G). \end{aligned}$$

■

Proof of Proposition 4 Let $F = \bigcup_{k=1}^{p} T_k$ and $G^*_{\mathcal{R}(F)}$ be as in the Proposition's hypotheses. Without loss of generality, we consider i to be the root of the tree T_i. Therefore, $\mathcal{R}(F) = [\![1, p]\!]$. Let us also define the following *root* function that associates to each node the root of the tree it belongs to:

$$r : \left| \begin{array}{ccc} V & \longrightarrow & \mathcal{R}(F) \\ i & \longmapsto & k \; s.t. \; X_i \in V_{T_k}. \end{array} \right.$$

Let $G^*_{\mathcal{R}(F)} \in DAG_{\mathcal{R}(F)}$ such that:

$$G^*_{\mathcal{R}(F)} \in \underset{G \in DAG_{\mathcal{R}(F)}}{\operatorname{argmax}} \; s_D^{MLL}(G)$$

and $G^* = F \cup G^*_{\mathcal{R}(F)}$ *i.e.*

- $V_{G^*} = V$
- $A_{G^*} = \left(\bigcup_{k=1}^{p} A_{T_k}\right) \cup A_{G^*_{\mathcal{R}(F)}}$

We will show as in the proof of Proposition 3 that

$$s^{MLL}_{DAG_V}(G^*) \geq \max_{G \in DAG_V} s^{MLL}_{DAG_V}(G)$$

which implies that $G^* \in \underset{G \in DAG_V}{\operatorname{argmax}} \; s^{MLL}_{DAG_V}(G)$.

We write:

$$\begin{aligned} s^L_{DAG_V}(G^*) &= -M \sum_{i=1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)}) \\ &= \underbrace{-M \sum_{i=1}^{p} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)})}_{(a)} \\ &\quad \underbrace{-M \sum_{i=p+1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)})}_{(b)} \end{aligned}$$

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

- *Computation of (a)*

  The first term corresponds to the score of the graph $G^*_{\mathcal{R}(F)}$ as an element of $DAG_{\mathcal{R}(F)}$. Indeed, by construction of G^*,

  $$\forall i \in \mathcal{R}(F),\ \pi^{G^*}(i) = \pi^{G^*_{\mathcal{R}(F)}}(i).$$

  Moreover, $G^*_{\mathcal{R}(F)}$ maximizes the MLL score on $DAG_{\mathcal{R}(F)}$. We can now write:

  $$\begin{aligned} (a) &= -M \sum_{i=1}^{p} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)}) \\ &= -M \sum_{i=1}^{p} H^D(X_i|\mathbf{X}_{\pi^{G^*_{\mathcal{R}(F)}}(i)}) \\ &= s^{MLL}_D(G^*_{\mathcal{R}(F)}) \\ &= \max_{G \in DAG_{\mathcal{R}(F)}} s^{MLL}_{D_{\mathcal{R}(F)}}(G) \\ &= -MH^D(X_1, \ldots, X_p). \end{aligned}$$

- *Computation of (b)*

  By construction of G^*,

  $$\forall i \in V \setminus \mathcal{R}(F),\ \pi^{G^*}(i) = \pi^{T_{r(i)}}(i).$$

  Moreover since the T_k's are deterministic trees, it follows that

  $$\forall i \in V \setminus \mathcal{R}(F),\ H^D(X_i|\mathbf{X}_{\pi^{T_{r(i)}}(i)}) = 0.$$

  Therefore we can write

  $$\begin{aligned} (b) &= -M \sum_{i=p+1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)}) \\ &= -M \sum_{i=p+1}^{n} H^D(X_i|\mathbf{X}_{\pi^{T_{r(i)}}(i)}) \\ &= 0. \end{aligned}$$

Collecting the above results yields

$$\begin{aligned} s^{L}_{DAG_V}(G^*) &= (a) \\ &= -MH^D(X_1, \dots, X_p) \\ &\geq -MH^D(X_1, \dots, X_n) \\ &= \max_{G \in DAG_V} s^{MLL}_D(G). \end{aligned}$$

■

Remarks after proof of Proposition 4

- The last inequality we wrote in the proof is of course an equality:

  $$H^D(X_1, \dots, X_p) = H^D(X_1, \dots, X_n)$$

  More generally, for $G \in DAG_V$ a deterministic DAG, and $\mathbf{X}$ a tuple of variables indexed by V,

  $$H^D(\mathbf{X}) = H^D(\mathbf{X}_{\mathcal{R}(G)}). \tag{15}$$

  In other words, the joint entropy of all the nodes in the graph is equal to the entropy of the root nodes only.

  Here is another way to prove it:
  For any random variables X, Y we have: $H^D(X|Y) \leq H^D(X)$, which notably implies that: $H^D(X,Y) \leq H^D(X) + H^D(Y)$.
  Moreover, we define the function 'root in G', R_G as follows:

  $$\forall i \in V - \mathcal{R}(G),\ R_G(i) = \mathcal{R}(G) \cap Ansc^G(i)$$

  We can then write:

  $$\begin{aligned} H^D(\mathbf{X}_{V \setminus \mathcal{R}(G)} \mid \mathbf{X}_{\mathcal{R}(G))} &\leq \sum_{i \in V \setminus \mathcal{R}(G)} \underbrace{H^D(X_i \mid \mathbf{X}_{\mathcal{R}(G)})}_{\leq H^D(X_i | X_{R_G(i)})} \\ &\leq \sum_{i \in V \setminus \mathcal{R}(G)} \underbrace{H^D(X_i | X_{R_G(i)})}_{=0} \\ &= 0 \end{aligned}$$

  which gives $H^D(\mathbf{X}_{V \setminus \mathcal{R}(G)} \mid \mathbf{X}_{\mathcal{R}(G)}) = 0$.
  Injecting this in the following equation:

  $$H^D(\mathbf{X}) = H^D(\mathbf{X}_{\mathcal{R}(G)}) + H^D(\mathbf{X}_{V \setminus \mathcal{R}(G)} \mid \mathbf{X}_{\mathcal{R}(G)})$$

  proves that Equation (15) holds.

Proof of Proposition 8 The structure of the proof is the same as the one from Proposition 2. The only difference lies in the computation of term (b):

$$\begin{aligned}(b) &= -M \sum_{i=p+1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G^*}(i)}) \\ &= -M \sum_{i=p+1}^{n} \underbrace{H^D(X_i|\mathbf{X}_{\pi^{T_{r(i)}}(i)})}_{\leq \varepsilon} \\ &\geq -M(n-p)\varepsilon \\ &\geq -Mn\varepsilon.\end{aligned}$$

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

Proof of Lemma 3 Let G and G' satisfy the conditions of the lemma:

$$\begin{aligned}s_D^{MLL}(G') - s_D^{MLL}(G) &= -M \sum_{i=1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G'}(i)}) - H^D(X_i|\mathbf{X}_{\pi^{G}(i)}) \\ &= -M \left(\underbrace{\sum_{i \neq i_1} H^D(X_i|\underbrace{\mathbf{X}_{\pi^{G'}(i)}}_{\mathbf{X}_{\pi^{G}(i)}}) - H^D(X_i|\mathbf{X}_{\pi^{G}(i)})}_{=0} \right) \\ &\quad - M \left(H^D(X_{i_1}|\underbrace{\mathbf{X}_{\pi^{G'}(i_1)}}_{\mathbf{X}_{\pi^{G}(i_1)} \cup \{X_{i_0}\}}) - H^D(X_{i_1}|\mathbf{X}_{\pi^{G}(i_1)}) \right) \\ &= -M \left(H^D(X_{i_1}|\mathbf{X}_{\pi^{G}(i_1)}, X_{i_0}) - H^D(X_{i_1}|\mathbf{X}_{\pi^{G}(i_1)}) \right) \\ &= M\hat{I}(X_{i_1}, X_{i_0}|\mathbf{X}_{\pi^{G}(i_1)})\end{aligned}$$

■

Proof of Lemma 4 Let $\sigma \in \mathcal{S}_n$.
For any $i \in \{1, \ldots, n\}$, by definition:

$$\mathbf{X}_{\pi^{G^\sigma}(i_1)} = \{X_j,\ \sigma^{-1}(j) < \sigma^{-1}(i)\}$$

Thanks to Lemma 2 we can write:

$$\begin{aligned}s_D^{MLL}(G^\sigma) &= -M \sum_{i=1}^{n} H^D(X_i|\mathbf{X}_{\pi^{G^\sigma}(i_1)}) \\ &= -M \sum_{i=1}^{n} H^D(X_i|X_j \ s.t.\ \sigma^{-1}(j) < \sigma^{-1}(i))\end{aligned}$$

We use the index switch $k = \sigma^{-1}(i)$, that gives us:

$$\begin{aligned} s_D^{MLL}(G^\sigma) &= -M \sum_{k=1}^{n} H^D(X_{\sigma(k)}|X_{\sigma(l)} \ st \ l < k) \\ &= -M \left(H^D(X_{\sigma(1)}) + H^D(X_{\sigma(2)}|X_{\sigma(1)}) + H^D(X_{\sigma(3)}|X_{\sigma(1)}, X_{\sigma(2)}) + \dots \right. \\ &\left. + H^D(X_{\sigma(n)}|X_{\sigma(1)}, \dots, X_{\sigma(n-1)}) \right) \end{aligned}$$

For any two random variables X, Y, the Bayes relation for entropies (still true for empirical entropies) gives that $H(X, Y) = H(X) + H(Y|X)$.
Using this relation, an immediate induction gives us:

$$\begin{aligned} s_D^{MLL}(G^\sigma) &= -MH^D(X_{\sigma(1)}, \dots, X_{\sigma(n)}) \\ &= -MH^D(X_1, \dots, X_n) \end{aligned}$$

■

Proof of Lemma 5 We have seen in Chapter 1 that, for any $G \in DAG_V$, there exists an ordering $\sigma_G \in S_n$ such that G is consistent with σ_G. For such a σ_G, we have that $A_G \subset A_{G_{comp}^{\sigma_G}}$, *i.e.* $G_{comp}^{\sigma_G}$ can be obtained by adding arcs to G.

Moreover, Lemma 3 implies that adding an arc to a DAG G will not decrease its MLL score. In other words, if we add arcs to G until it becomes 'complete', we will not decrease its MLL score.
Finally, Lemma 4 implies that $s_D^{MLL}(G^\sigma)$ does not depend on σ for $\sigma \in S_n$, and that

$$\forall \sigma \in S_n, \ s_D^{MLL}(G^\sigma) = -MH^D(X_1, \dots, X_n)$$

Combining this, we obtain that for any G in DAG_V, if $\sigma_G \in S_n$ is an ordering such that G is consistent with σ_G, we have:

$$\begin{aligned} s_D^{MLL}(G) &\leq s_D^{MLL}(G^{\sigma_G}) \\ &= -MH^D(X_1, \dots, X_n) \end{aligned}$$

This notably gives that

$$\max_{G \in DAG_V} s_D^{MLL}(G) = -MH^D(X_1, \dots, X_n)$$

■

Proof of Lemma 6 By definition, a forest $F = (V, A)$ is the union of disjoint trees $\{T_k\}$ (in the sense of the graphical union). Since F is deterministic, every node $i \in V$ that has at least one parent, *i.e.* that is not one of the tree roots, corresponds to a variable X_i such that

$$H^D(X_i|X_{\pi^F(i)}) = 0.$$

Since $pi^F(i)$ belongs to the same tree as i, we get that for any given tree T_k, all the variables that correspond to a node that is not the roots of the tree is entirely determined by its parents **in the tree**: T_k is a deterministic DAG, and therefore a deterministic tree. ■

Proof of Lemma 7 The proof is very straightforward. We just have to write two different ways the empirical mutual information of X_i and X_j, and use the fact that it is symmetric. Let us suppose that $X_i \underset{D}{\leftrightarrow} X_j$, we can write:

$$\begin{aligned} I^D(X_i; X_j) &= H^D(X_i) - \underbrace{H^D(X_i|X_j)}_{=0} \\ &= H^D(X_i) \\ I^D(X_j; X_i) &= H^D(X_j) - \underbrace{H^D(X_j|X_i)}_{=0} \\ &= H^D(X_j). \end{aligned}$$

Moreover, $I^D(X_j; X_i) = I^D(X_i; X_j)$, therefore we indeed have that $H^D(X_i) = H^D(X_j)$.
■

Proof of Lemma 8 If $H^D(X_i|X_j) = 0$, we have seen that there exists a function

$$f : Val(X_j) \to Val(X_i)$$

such that for all $x_i \in Val(X_j)$ and $x_j \in Val(X_i)$,

$$p^D(x_i|x_j) = \mathbb{I}_{x_i = f(x_j)}.$$

This function f is **surjective**: for $x_i \in Val(X_i)$, we know x_i appears at least once in D (otherwise we would not have considered it to be in $Val(X_i)$, as explained in Section 1.1.2 of Chapter 2).
Let m be a row on which x_i appears in D. Since the dataset is complete, we know that

$$x_i = f(x_j^{(m)}).$$

This indeed proves that f is surjective, which directly yields:

$$|Val(X_i)| \leq |Val(X_j)|.$$

Moreover, if $|Val(X_i)| = |Val(X_j)|$, then f is bijective and admits an inverse function f^{-1}, which is defined in $Val(X_i)$ and has values in $Val(X_j)$.
Using the fact that $p^D(x_j|x_i) = \frac{p^D(x_i|x_j)p^D(x_j)}{p(x_i)}$, we immediately get that for all $x_j \in Val(X_i)$ and $x_i \in Val(X_j)$,

$$p^D(x_j|x_i) = \mathbb{I}_{x_j = f^{-1}(x_1)}.$$

The relationship $X_i \to X_j$ is therefore *empirically* functional, which gives that $H^D(X_j|X_i) = 0$. Therefore, $X_i \underset{D}{\leftrightarrow} X_j$. The other implication is trivial. ■

Proof of Proposition 5 We show that $\underset{D}{\leftrightarrow}$ is an equivalence relationship on $\{X_1, \ldots, X_n\}$. Indeed, the binary relation $\underset{D}{\leftrightarrow}$ is:

1. **reflexive**: $\forall i$, we obviously have $H^D(X_i|X_i) = 0$,
2. **symmetric**: $\forall i, j$, X_i and X_j play symmetric roles in Definition 6,
3. **transitive**: Suppose that for $i, j, k \in \{1, \ldots, n\}$, we have $X_i \underset{D}{\leftrightarrow} X_j$ and $X_j \underset{D}{\leftrightarrow} X_k$. Then

$$\begin{aligned}
H^D(X_i, X_j, X_k) &= H^D(X_i, X_k|X_j) + H^D(X_j) \\
&= \underbrace{H^D(X_i|X_j, X_k)}_{\leq H^D(X_i|X_j)=0} + \underbrace{H^D(X_k|X_j)}_{=0} + H^D(X_j) \\
&= H^D(X_j) \\
H^D(X_i, X_j, X_k) &= H^D(X_i, X_j|X_k) + \underbrace{H^D(X_k)}_{=H^D(X_j)} \\
&= \underbrace{H^D(X_j|X_i, X_k)}_{\leq H^D(X_j|X_i)=0} + H^D(X_i|X_k) + H^D(X_j) \\
&= H^D(X_i|X_k) + H^D(X_j) \\
H^D(X_i, X_j, X_k) &= H^D(X_k, X_j|X_i) + \underbrace{H^D(X_i)}_{=H^D(X_j)} \\
&= \underbrace{H^D(X_j|X_k, X_i)}_{\leq H^D(X_j|X_k)=0} + H^D(X_k|X_i) + H^D(X_j) \\
&= H^D(X_k|X_i) + H^D(X_j)
\end{aligned}$$

Using these three relations, we get $H^D(X_i|X_k) = H^D(X_k|X_i) = 0$, *i.e.* $X_i \underset{D}{\leftrightarrow} X_k$,

Proof of Proposition 7 By definition, such a graph G is *directed*, and every node has *at most one parent*. Moreover, the fact that there is no redundant variable w.r.t. D proves that there cannot be any cycle in G (reciprocal of Proposition 6). Therefore G is *acyclic*.
This indeed proves that G is a *forest*, by definition. ■

3 Proofs of results presented in Chapter 4

Proof of Lemma 9 This proof is simply a technical rewriting. The lemma is not that useful per se, just as an intermediate result for Proposition 9.

We remind that for $\Theta \in \vartheta_G$, p_Θ factorizes in G, *i.e.* for all $(x_1, \dots, x_n) \in Val(\mathbf{X})$, we have

$$p(x_1, \dots, X_n) = \prod_{i=1}^{n} p(x_i | \mathbf{x}_{\pi^G(i)}).$$

We may then write:

$$\begin{aligned}
\mathcal{H}(p^* || p_\Theta) &= \sum_{(x_1,\dots,x_n) \in Val(X_1)\times\cdots\times Val(X_n)} p^*(x_1, \dots, x_n) \log(p_\Theta(x_1, \dots, x_n)) \\
&= \sum_{(x_1,\dots,x_n) \in Val(X_1)\times\cdots\times Val(X_n)} p^*(x_1, \dots, x_n) \sum_{i=1}^{n} \log(p_\Theta(x_i | \mathbf{x}_{\pi(i)})) \\
&= \sum_{(x_1,\dots,x_n) \in Val(X_1)\times\cdots\times Val(X_n)} \sum_{i=1}^{n} p^*(x_1, \dots, x_n) \log(p_\Theta(x_i | \mathbf{x}_{\pi(i)})) \\
&= \sum_{i=1}^{n} \sum_{(x_1,\dots,x_n) \in Val(X_1)\times\cdots\times Val(X_n)} p^*(x_1, \dots, x_n) \log(p_\Theta(x_i | \mathbf{x}_{\pi(i)})) \\
&= \sum_{i=1}^{n} \sum_{(x_i, \mathbf{x}_{\pi(i)}) \in Val(X_i)\times Val(\mathbf{X}_{\pi(i)})} \underbrace{\sum_{x_j \in Val(X)_j \mid X_j \notin \{X_i, \mathbf{X}_{\pi(i)}\}} p^*(x_1, \dots, x_n)}_{p^*(x_i, \mathbf{x}_{\pi(i)})} \log(p_\Theta(x_i | \mathbf{x}_{\pi(i)})) \\
&= \sum_{i=1}^{n} \sum_{(x, \mathbf{x}_{\pi(i)}) \in Val(X_i)\times Val(\mathbf{X}_{\pi(i)})} p^*(x, \mathbf{x}_{\pi(i)}) \log(p_\Theta(x | \mathbf{x}_{\pi(i)}))
\end{aligned}$$

■

Proof of Proposition 9 For any Θ defining a distribution of $\mathbf{X}$,

$$\begin{aligned}
\mathcal{H}(p_D || p_\Theta) &= \sum_{i=1}^{n} \sum_{x,u \in Val(X_i)\times Val(\mathbf{X}_{\pi(i)})} p_D(x, \mathbf{x}_{\pi(i)}) \log(p_\Theta(x | \mathbf{x}_{\pi(i)})) \\
&= \sum_{i=1}^{n} \sum_{x,u \in Val(X_i)\times Val(\mathbf{X}_{\pi(i)})} \frac{C^D(x, \mathbf{x}_{\pi(i)})}{M} \log(p_\Theta(x | \mathbf{x}_{\pi(i)})) \\
&= \frac{1}{M} \sum_{i=1}^{n} \sum_{x,u \in Val(X_i)\times Val(\mathbf{X}_{\pi(i)})} C^D(x, \mathbf{x}_{\pi(i)}) \log(p_\Theta(x | \mathbf{x}_{\pi(i)})) \\
&= -\frac{1}{M} l_D(p_\Theta).
\end{aligned}$$

Therefore, using the fact that for two distributions p, q, we have the relation:

$$D_{KL}(p||q) = \mathcal{H}(p) - \mathcal{H}(p||q),$$

we immediatly get that:

$$\underset{\Theta \in \vartheta_G}{\operatorname{argmin}} D_{KL}(p^D || p_\Theta) = \underset{\theta \in \vartheta_G}{\operatorname{argmax}} l_D(\Theta)$$

■

Proof of Lemma 10 The validation likelihood (with a training set T and a validation set V), presented in Section 3.4.2, is defined by:

$$s^{VLL}_{T,V}(G) = l_V(\hat{\Theta}^T) \tag{16}$$

Now, thanks to the definition given in Equation (16), the validation-likelihood score for a structure G can be written as:

$$\begin{aligned} s^{VLL}_{T,V}(G) &= -\sum_{i=1}^{n} \sum_{(x,\mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi(i)})} C^V(x, \mathbf{x}_{\pi(i)}) \log\left(\hat{\theta}^T_{x|\mathbf{x}_{\pi(i)}}\right) \\ &= -M_V \sum_{i=1}^{n} \sum_{(x,\mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi(i)})} \frac{C^V(x, \mathbf{x}_{\pi(i)})}{M_V} \log\left(\hat{\theta}^T_{x|\mathbf{x}_{\pi(i)}}\right) \\ &= -M_V \sum_{i=1}^{n} \sum_{(x,\mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi(i)})} \hat{\theta}^V_{x,\mathbf{x}_{\pi(i)}} \log\left(\hat{\theta}^T_{x|\mathbf{x}_{\pi(i)}}\right) \end{aligned}$$

Where $\hat{\Theta}^V$ is simply the distribution that would have been learned by training the model on the validation set V.

Lemma 9 gives that, given a categorical random variable X and for any $p, q \in \mathcal{F}^P(Val(X))$ such that q factorizes in G, we can write:

$$\mathcal{H}(p||q) = \sum_{i=1}^{n} \sum_{(x_i,\mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi(i)})} p(x_i, \mathbf{x}_{\pi(i)}) \log\left(q(x_i|\mathbf{x}_{\pi(i)})\right).$$

Applying this to $p = p_{\hat{\Theta}^V}$ and $q = p_{\hat{\Theta}^T}$, and reminding that for $\Theta \in \vartheta_G$, we have the notation

$$\theta_{x_i|\mathbf{x}_{\pi(i)}} = p_\Theta(x_i|\mathbf{x}_{\pi(\mathbf{i})}),$$

we get:

$$\sum_{i=1}^{n} \sum_{(x,\mathbf{x}_{\pi(i)}) \in Val(X_i) \times Val(\mathbf{X}_{\pi(i)})} \hat{\theta}^V_{x,\mathbf{x}_{\pi(i)}} \log\left(\hat{\theta}^T_{x|\mathbf{x}_{\pi(i)}}\right) = \mathcal{H}(p_{\hat{\Theta}^V} || p_{\hat{\Theta}^T})$$

Therefore, we can finally write

$$s^{VLL}_{T,V}(G) = -M_V \mathcal{H}(p^V_{\hat{\Theta}} || p^T_{\hat{\Theta}})$$

■

Proof of Proposition 10 The proof simply consists on rewriting Equation (4.7) using Equation (4.1). ■

Additional information for figures

Table 2: Dataset D_1: legend for Figure 2.4 in Section 5.2.2 of Chapter 2

Var. Ref.	Variable Name	Nb. Levels
X1	DATA_binarystate_name	2
X2	DATA_binarystate_state0	2
X3	DATA_binarystate_state1	2
X4	DATA_data_id	922
X5	DATA_data_name	628
X6	DATA_data_offset_applied	2
X7	DATA_data_scalefactor_applied	3
X8	DATA_datatype_code	18
X9	DATA_datatype_name	18
X10	DATA_datatype_name_fr	18
X11	DATA_factsgroup_name	6
X12	DATA_KEY	1000
X13	DATA_metafactsgroup_name	4
X14	DATA_unit_name	11
X15	DATA_unit_name_fr	11
X16	DATA_unit_symbol	11
X17	DATASOURCE_datacollector_factsgroup_name	6
X18	DATASOURCE_datacollector_factsgroup_name_dds	6
X19	DATASOURCE_datacollector_metafactsgroup_name	4
X20	DATASOURCE_datacollector_name	736
X21	DATASOURCE_datacollector_nb_datadescriptions_handled	5
X22	DATASOURCE_datacollector_samplingperiod_minutes	3
X23	DATASOURCE_datacollector_shortname	762
X24	DATASOURCE_dataprovider_name	3
X25	DATASOURCE_KEY	762
X26	DATASOURCE_sensor_name	345
X27	DATASOURCE_sensor_shortname	327
X28	DATASOURCE_sensormodel_comment	4
X29	DATASOURCE_sensormodel_manufacturer	7
X30	DATASOURCE_sensormodel_name	15
X31	DATASOURCE_sensormodel_shortname	15
X32	SPACE_dataprovider_site	10
X33	SPACE_dataprovider_zonecode	109
X34	SPACE_KEY	151
X35	SPACE_site_name	10
X36	SPACE_site_surface_shab	8
X37	SPACE_site_surface_shon	5
X38	SPACE_sitetype_code	5
X39	SPACE_sitetype_name	5
X40	SPACE_sitetype_name_fr	5
X41	SPACE_zone_code	75
X42	SPACE_zone_name_en	109
X43	SPACE_zone_name_fr	106
X44	SPACE_zone_surface	38
X45	SPACE_zonetype_code	25
X46	SPACE_zonetype_name	25
X47	SPACE_zonetype_name_fr	25

Table 3: Dataset D_2: legend for Figure 2.4 in Section 5.2.2 of Chapter 2

Var. Reference	Variable Name	Nb. Levels
X1	DATA_binarystate_name	2
X2	DATA_binarystate_state0	2
X3	DATA_binarystate_state1	2
X4	DATA_data_name	628
X5	DATA_data_offset_applied	2
X6	DATA_data_scalefactor_applied	3
X7	DATA_datatype_code	18
X8	DATA_datatype_name	18
X9	DATA_datatype_name_fr	18
X10	DATA_factsgroup_name	6
X11	DATA_metafactsgroup_name	4
X12	DATA_unit_name	11
X13	DATA_unit_name_fr	11
X14	DATA_unit_symbol	11
X15	DATASOURCE_datacollector_factsgroup_name	6
X16	DATASOURCE_datacollector_factsgroup_name_dds	6
X17	DATASOURCE_datacollector_metafactsgroup_name	4
X18	DATASOURCE_datacollector_name	736
X19	DATASOURCE_datacollector_nb_datadescriptions_handled	5
X20	DATASOURCE_datacollector_samplingperiod_minutes	3
X21	DATASOURCE_dataprovider_name	3
X22	DATASOURCE_sensor_name	345
X23	DATASOURCE_sensor_shortname	327
X24	DATASOURCE_sensormodel_comment	4
X25	DATASOURCE_sensormodel_manufacturer	7
X26	DATASOURCE_sensormodel_name	15
X27	DATASOURCE_sensormodel_shortname	15
X28	SPACE_dataprovider_site	10
X29	SPACE_dataprovider_zonecode	109
X30	SPACE_KEY	151
X31	SPACE_site_name	10
X32	SPACE_site_surface_shab	8
X33	SPACE_site_surface_shon	5
X34	SPACE_sitetype_code	5
X35	SPACE_sitetype_name	5
X36	SPACE_sitetype_name_fr	5
X37	SPACE_zone_code	75
X38	SPACE_zone_name_en	109
X39	SPACE_zone_name_fr	106
X40	SPACE_zone_surface	38
X41	SPACE_zonetype_code	25
X42	SPACE_zonetype_name	25
X43	SPACE_zonetype_name_fr	25

Table 4: Dataset D_3: legend for Figure 2.4 in Section 5.2.2 of Chapter 2

Var. Reference	Variable Name	Nb. Levels
X1	DATA_binarystate_name	2
X2	DATA_binarystate_state0	2
X3	DATA_binarystate_state1	2
X4	DATA_data_offset_applied	2
X5	DATA_data_scalefactor_applied	3
X6	DATA_datatype_code	18
X7	DATA_datatype_name	18
X8	DATA_datatype_name_fr	18
X9	DATA_factsgroup_name	6
X10	DATA_metafactsgroup_name	4
X11	DATA_unit_name	11
X12	DATA_unit_name_fr	11
X13	DATA_unit_symbol	11
X14	DATASOURCE_datacollector_factsgroup_name	6
X15	DATASOURCE_datacollector_factsgroup_name_dds	6
X16	DATASOURCE_datacollector_metafactsgroup_name	4
X17	DATASOURCE_datacollector_nb_datadescriptions_handled	5
X18	DATASOURCE_datacollector_samplingperiod_minutes	3
X19	DATASOURCE_dataprovider_name	3
X20	DATASOURCE_sensor_name	345
X21	DATASOURCE_sensor_shortname	327
X22	DATASOURCE_sensormodel_comment	4
X23	DATASOURCE_sensormodel_manufacturer	7
X24	DATASOURCE_sensormodel_name	15
X25	DATASOURCE_sensormodel_shortname	15
X26	SPACE_dataprovider_site	10
X27	SPACE_dataprovider_zonecode	109
X28	SPACE_KEY	151
X29	SPACE_site_name	10
X30	SPACE_site_surface_shab	8
X31	SPACE_site_surface_shon	5
X32	SPACE_sitetype_code	5
X33	SPACE_sitetype_name	5
X34	SPACE_sitetype_name_fr	5
X35	SPACE_zone_code	75
X36	SPACE_zone_name_en	109
X37	SPACE_zone_name_fr	106
X38	SPACE_zone_surface	38
X39	SPACE_zonetype_code	25
X40	SPACE_zonetype_name	25
X41	SPACE_zonetype_name_fr	25