THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Santiago José CORTIJO ARAGON

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Sujet de la thèse :

Sécurité pour des infrastructures critiques SCADA fondée sur des modèles graphiques probabilistes

(Probabilistic graphical model-based security for SCADA critical infrastructures)

soutenue le 10 décembre 2018

devant le jury composé de:

Directeur de thèse:	M. Christophe GONZALES *Professeur* à Sorbonne Université.
Rapporteurs:	M. Alexandre AUSSEM *Professeur* à l'Université Lyon 1. M. Philippe LERAY *Professeur* à l'Université de Nantes.
Examinateurs:	Mme. Véronique DELCROIX *Maître de Conférences* à l'UPHF. M. Nicolas MAUDET *Professeur* à Sorbonne Université. M. CédricTAVERNIER *Docteur* à ASSYSTEM.

Acknowledgments

I would like to thank Christophe Gonzales, my thesis director, for trusting me to follow a PhD, which was by far the most enriching academic experience I ever had. He was always involved in my research work and was extremely supportive, specially in the most critical moments of the last three years. This thesis work would have been certainly unfeasible without his support.

I must also mention Pierre-Henri Wuillemin who, without having any direct responsibility for my work, took plenty of his own time to give me high quality technical advice. I also must mention that he was the one that arranged the funding for my last three months of PhD work; without his help I would have not been able to have defend this thesis.

I would also like to thank Thibaut Lust, who gave me the opportunity to follow an internship at LiP6, which I found to be a nice and healthy environment to do my research work, this experience led me to apply to the to this PhD at LIP6. I must also thank the SCISSOR project members, for giving me the priceless experience of interacting with people of different nationalities at once, and specially to Lionel Torti for giving me good advice.

I am deeply grateful to my thesis reporters Alexandre Aussem et Philippe Leray, for taking many hours of their precious time to read this manuscript in detail and to give many suggestions and remarks for improving it and as well as for being part of my defense jury. Véronique Delcroix, Cédric Tavernier, Nicolas Maudet for taking from their time to participate in my thesis defense as members of jury.

I also want to thank the permanent members of LIP6 Olivier Spanjaard, Christoph Dürr, Bruno Escoffier, Pierre Fouilloux, Paolo Viappiani Safia Kedad-Sidhoum, Viet Hung Nguyen, Fanny Pascual and Patrice Perny; and to my PhD fellows Ahmed Mabrouk, Hamza Agli, Marvin Lasserre, Gaspard Ducamp, Siao-Leu Phouratsamay, Hugo Gilbert, Franco Quezada, Matthieu Hourbracq, Nawal Benabbou, Morgan Chopin, Cécile Rottner, Nadjet Bourdache, Anne-Elisabeth Falq, Adèle Pass-Lanneau, Alexandre Teillier, Parham Shams and Hugo Martin.

My final and warmest thanks are to my deeply beloved family: José, Maritza, Elsa and Dounia.

Contents

Common Abbreviations

BN	Bayesian network.
CCL	Coordination & Control Layer (SCISSOR project).
CLG	Conditional linear Gaussians.
CPT	Conditional probability table.
CdBN	Conditional densities BN.
CtdBN	Conditional truncated densities BN.
DAG	Directed acyclic graph.
DAL	Decision & Analysis Layer (SCISSOR project).
DBN, ns-DBN	Dynamic BN, non-stationnary DBN.
HMI	Human-Machine Interface (SCISSOR project).
JT	Junction tree.
ML	Monitoring Layer.
MTBF	Mixture of truncated basis functions.
MTE	Mixture of truncated exponentials.
MOP	Mixture of truncated polynomials.
SCADA	Supervisory control and data acquisition system.
SCISSOR	Security in trusted SCADA and smart-grids.
SIEM	Security Informayion & Event Management (SCISSOR project).
SS	Shafer-Shenoy.
TDM	Threat detection module (SCISSOR project).
VE	Variable elimination.

Notations

$P(\cdot), f(\cdot)$	A probability distribution (resp. density function).
$P(\cdot \mid \cdot), f(\cdot \mid \cdot)$	A conditional probability (resp. density).
ϕ, Φ	A factor (resp. factor set).
X, Y, Z	Discrete random variables.
$\mathring{X}, \mathring{Y}, \mathring{Z}$	Continuous random variables.
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	Random variables sets.
$x, y, z; \mathbf{x}, \mathbf{y}, \mathbf{z}$	Instantiations of X, Y, Z; $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, respectively.
$\Omega_X, \Omega_{\mathbf{X}}$	Domains of X (resp. $\mathbf{X}$).
$(X \perp\!\!\!\perp Y)$	X is independent of Y.
$(X \perp\!\!\!\perp Y \mid Z)$	X is conditionally independent of Y given Z.
$P \models (X \perp\!\!\!\perp Y \mid Z)$	P models the independence $(X \perp\!\!\!\perp Y \mid Z)$.
$\mathcal{K} = (\mathbf{V}, \mathcal{E})$	A graph over nodes $\mathbf{V}$ and edges $\mathcal{E}$.
$\mathcal{H}, \mathcal{G}$	An undirected (resp. directed) graph.
$\mathbf{Pa}_{\mathcal{G}}(X)$, $\mathbf{Ch}_{\mathcal{G}}(X)$	Set of Parents (resp. Children) of X in a graph $\mathcal{G}$.
$\mathbf{MB}_{\mathcal{G}}(X)$	Markov Blanket of X in a graph $\mathcal{G}$.
$\mathbf{Ngb}_{\mathcal{H}}(X)$	Neighbors of X in a graph $\mathcal{H}$.
$X - Y$	An undirected edge (in a graph).
$X \to Y$, $X \leftarrow Y$, $X \leftrightarrow Y$	A directed edge (in a graph).
$X \rightleftharpoons Y$	A graph edge (directed or undirected).

Introduction

Critical Infrastructures are essential systems of assets, considered of vital importance for the correct functioning of society and economy: e.g. water supply, transportation, agriculture, telecommunications, electricity generation, etc. This definition may sound imprecise, and it actually varies from country to country. Nevertheless, there is a (common sense) consensus about the importance of protecting critical infrastructures against disruption or destruction, which globally increased after the terrorist attacks of September 11, 2001[1]. Critical infrastructures are often industrial facilities, and as such, they centralize their current state measurements in a single system which allows human users to perform remote actions (and sometimes it is programmed to perform pertinent actions by itself). A system with this qualities is called Supervisory Control and Data Acquisition System (SCADA, for short).

SCADA systems consist of hardware, software and communications channels linking their components. Figure 1 shows the general layout of the components of a SCADA system, which for simplicity displays only its basic components: There is a control center which performs centralized monitoring and control tasks, including monitoring alarms and processing/logging current status data, allowing human users to interact with the system through Human-Machine Interfaces (HMIs) and sometimes having enough autonomy to generate action upon certain events. The control center acquires data and controls the whole system through network connections coming from field devices such as Remote Telemetry Units (RTUs), Programmable Logic Controllers (PLCs) and Intelligent Electronic Devices (IEDs, e.g. as smart-cameras), which are the ones that directly acquire data from sensors and control actuators.

In the past, SCADA systems were typically isolated into industrial facilities, buy nowadays they have many of their components connected to the external world with Internet-base protocols such as TCP/IP, Modbus and other proprietary

[1]Refer to [Galland, 2010] for a discussion about different definitions of critical infrastructures across the countries)

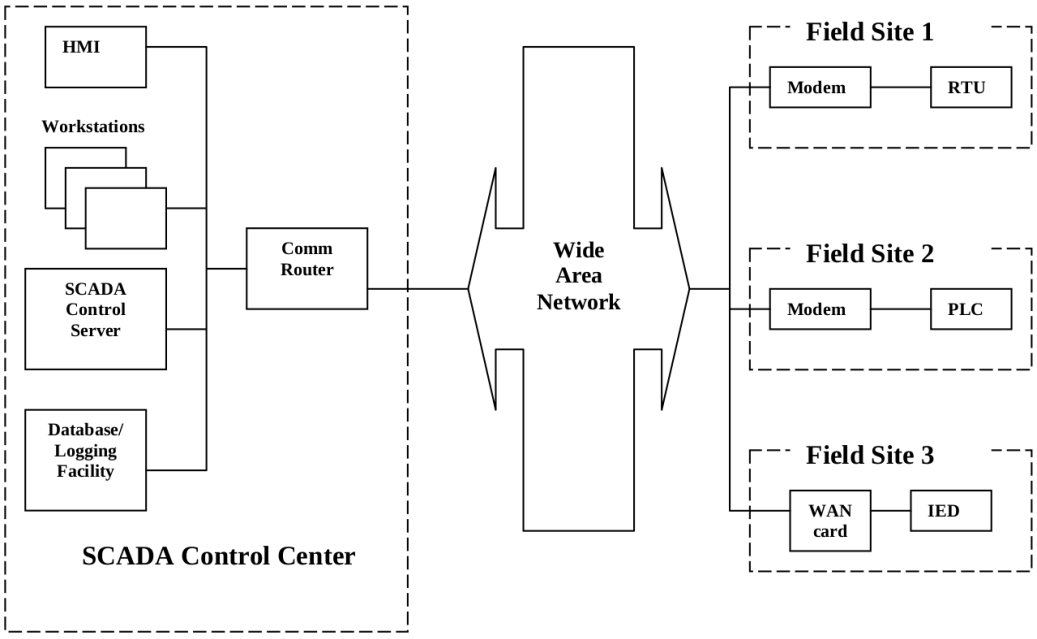


Figure 1: Simplified layout of a SCADA system. Image taken from [Tsang, 2010]

ones. The multiplicity and discrepancy of these highly interconnected components opens many opportunities to cyber-attack such systems. This even gives rise to the development cyber-weapons. For instance, STUXNET is a computer worm allegedly conceived to compromise Iranian SCADA nuclear systems [Karnouskos, 2011]. Discovered in 2010, it succeeded to sabotage Iran's nuclear program by interfering with the correct functioning of computers and PLCs. The final purpose of Stuxnet was to accelerate uranium enrichment centrifuges to the point of tearing them down. More recently, in 2015, there was a major cyberattack to a power grid in Ukraine [Lee et al., 2016] causing a blackout which affected about $230\,000$ people for periods between one and six hours.

Attacks like the aforementioned showed the limitations of the state-of-the-art SCADA security measures, which typically relied on proprietary technologies, often under the paradigm of "security by obscurity". This might give good results in most of the cases when the technology providers are trustworthy. However it disallows each societies' right to have full sovereignty over the systems governing its critical infrastructures. Having that in mind, through its H2020[2] program, the European community considered important to fund research on alternative SCADA protection frameworks, based on open protocols and open source tools. This gave

[2]https://ec.europa.eu/programmes/horizon2020/

rise to the SCISSOR project[3], of which this thesis is part.

The SCISSOR framework is structured into four layers, and its simplified view is presented in Figure 5.2, where we see its layers from bottom to top:

1. A monitoring layer (ML), where there are sensors and actuators components of the system,
2. A control and coordination layer (CCL), for grouping element of the ML in a pertinent way into edge agents, in order to collect all monitoring data in a uniform representation.
3. A decision & analysis layer (DAL), which receives the processed CCL data, and performs statistical/rule-based analysis.
4. A human-machine interface layer (HMI) presenting the system's behavior in real time in a human-readable manner, and displaying all the alerts that might come from the DAL or the CCL.

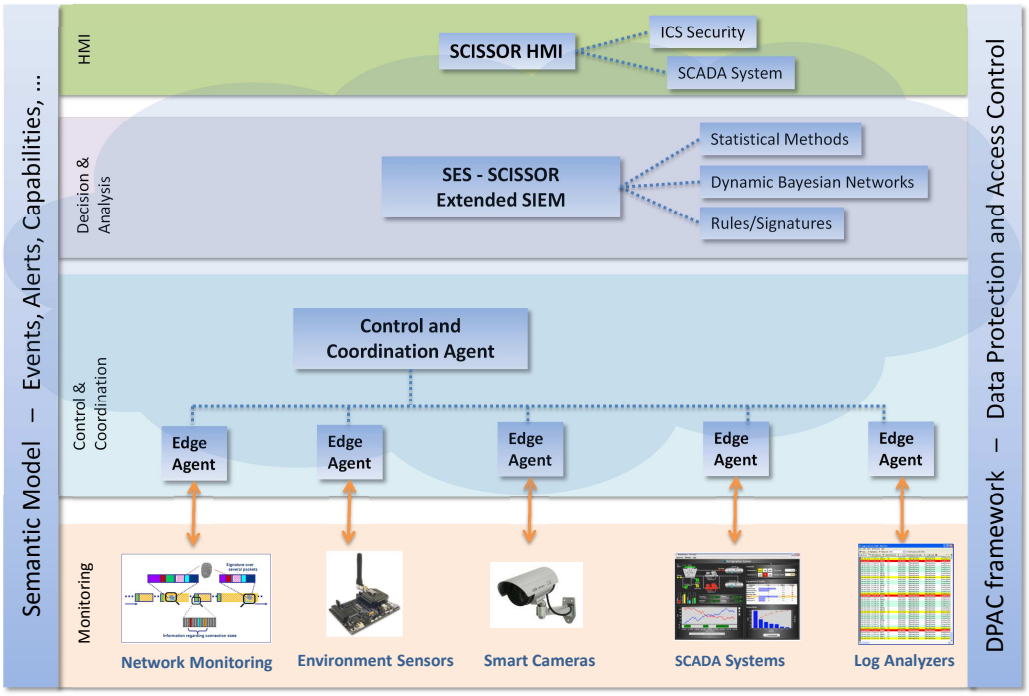


Figure 2: Overall architecture of SCISSOR system

The SCISSOR framework focuses on many different security aspects: preventing unauthorized physical access to industrial facilities, protecting its communication channels through firewalls and cryptography, verifying its log files

[3]https://scissor-project.com/

integrity (avoiding modifications/injections/deletions), testing extreme or unusual conditions in its field components, and statistically analyzing the whole state of the system. This last aspect is developed in Workpackage 5 of the SCISSOR project, in which this thesis is circumscribed.

The contribution of the present work is the use of Probabilistic Graphical Models (PGMs) to detect possible threats to SCADA systems. There exist several PGMs frameworks to deal with this kind of problem. We chose to base our work on Bayesian Networks (BNs), which have been successfully exploited in decision aid problems and diagnosis in many fields (medicine, robotics, computer troubleshooting, etc). BNs are well suited for SCISSOR purposes because they can model uncertainty in complex systems such as the SCADA ones, where there could be the necessity to perform inference under missing data conditions. In addition, unlike black-box models such as neural networks or SVMs, they have a semantically clear representation, which can be interpreted, corrected or even completely specified by experts.

The most widely used BN model is the one that deals with discrete/categorical random variables (the reader can refer to Chapter 1 for discrete BN foundations, definition, learning/inference algorithms as well as temporal extensions). However SCADA systems (as many others) contain also continuous variables. The simplest solution to deal with continuous variables consists in discretizing these variables, but this represents a great loss of information in the model w.r.t. to the real system, especially if we consider that the domain size of those discretized variables must remain relatively small (about five or six discretization intervals at most, with today's technology computers) in order to keep learning and inference tasks efficient and effective. In the literature there are some BN models that can deal with continuous variables like Conditional Linear Gaussians (CLGs) and Mixtures of Truncated Basis Functions (MTBFs); the former are easy to learn but are not always expressive enough (they always represent a joint multivariate normal distribution, and they struggle representing a discrete random variable depending of a continuous one), and the latter are very expressive thanks to the use of a large family of probability distributions, but their inference times are prohibitive because they involve computing a number of algebraic expressions that grows exponentially with the number of "factors" involved in the MTBF. For more details about BN discretization, CLG, MTBFs, the reader might refer to Chapter 2.

In the present work we address the continuous random variables issues in the BN context by introducing two new BN-based models (which are presented in Chapters 3 and 4). They are more expressive than CLGs and scale up much better in terms of inference times than MTBFs. In the last thirty years, there was a lot

of research efforts in the learning and inference of discrete BNs (which are NP-complete problems); we propose adaptations of some of those algorithms for our proposed models.

Keeping in mind that the main purpose of the SCISSOR framework is cyber-security, we could imagine various possible cyber-attacks scenarios of a SCADA system (some are actually documented in the literature). However, it is impossible to devise all the possible scenarios. In addition, cyberattacks are rare events that do not necessarily follow a unique pattern. So, it is difficult to define a statistical model of SCADA cyber-atacks. Rather, one can address the problem as as an anomaly detection problem, where a potential attack is defined as a very unlikely state in the system. This is the approach taken in this thesis. In Chapter 5, this approach is presented through testbeds on a real SCADA system found in the SEA FAVIGNANA[4], accompanied by a non-exhaustive description of the SCISSOR project itself.

Finally, the manuscript ends with a chapter presenting some conclusions and future research directions.

[4]An electric central in Favignana's island (Sicily, Italy). `https://www.seafavignana.com/`

Part I

State of the Art

Chapter 1

Bayesian Networks

After decades of research, multivariate uncertainty modeling is still being a challenge for the AI and Machine Learning communities. The first successful attempt to tackle this problem came with the introduction of Probabilistic Graphical Models (PGMs), which are declarative representations, *i.e.* they separate the knowledge they encode and the methods we use to reason through them; this allows using different algorithms for the same reasoning task, and using the same reasoning algorithms for different models. For instance, we are able to use the same inference algorithms for Bayesian Networks and Markov Networks (two different PGMs).

One of the first appearances of a PGM as such came with the introduction of the Bayesian Network model (BN, *aka.* Belief networks) [Pearl, 1988]. This thesis focuses on BN based models, for which we borrow and adapt BN learning and inference methods.

BNs are graphs whose nodes represent random variables and whose edges represent conditional dependencies. Although we will give a more rigorous statement of what a BN is, it is already worthy to mention that BNs have some advantages w.r.t. other state-of-the-art multivariate reasoning models (such as Neural Networks [McCulloch and Pitts, 1943], Fuzzy Logic models [Zadeh, 1965], Support Vector Machines [Cortes and Vapnik, 1995], and any *blackbox* model in general), which can be summarized in three points:

1. The BN reasoning is deeply founded in statistics and probabilities.
2. The BN representation has a clear interpretation so that an expert of a given domain can suggest corrections for a BN or even propose a completely new one, so that we can have a BN model even if we cannot find enough data to learn its parameters.

3. The BN representation does not discriminate between input and output variables. BNs allows us to query about any subset of its variables w.r.t. any information available.

BNs were successfully used in medical expert systems [Heckerman et al., 1992, Andreassen et al., 1992], classification problems [Friedman, 1997, Cheng and Greiner, 2013], spam filtering [Sahami et al., 1998, Pantel and Lin, 1998], risk management [Fenton and Neil, 2012, Kindermann et al., 1980], automatic diagnosis [Bottone et al., 2008, Cheng et al., 2013], troubleshooting [Skaanning et al., 2000, Huang et al., 2014], bioinformatics [Friedman et al., 2000, Murphy and Mian, 1999], robotics [Lebeltel et al., 2004, Schumann et al., 2012], and many others.

In this chapter, in a first section we give a brief introduction to the foundations of BNs: probability and graphs; then in the next section we define formally BNs. In the last two sections we discuss inference and learning in the context of BNs. Most of the concepts in this chapter came from [Koller and Friedman, 2009], to which the reader should refer for further concepts.

1.1 Statistics Foundations

In this section we make a brief introduction of how to express uncertainty through statistical concepts, notably through probabilities. Uncertainty is an inherent part of our lives: we cannot know exactly our weight, the current temperature, how much time will it take us to arrive to work in the morning, when are we getting sick, etc. We are used to not have perfect information about our environment, to have only clues or even beliefs about the state of the real world around us, and we are even used to make relevant decisions in such conditions. That is why it is important to define a formal way to express, measure and reason about all these uncertainties.

A very common way to express uncertainty is through probabilities. Probabilities are a numerical measures of how likely an event is to happen, in a given context; we state it formally through Definitions 1.1.1 and 1.1.2.

Definition 1.1.1 (Event) *An Event is any possible (or impossible) outcome of a given experiment. If an event is sure to happen, we call it a* certain *event; in the same logic, if an event will never happen, we say it is an* impossible *event (denoted as $\emptyset$). We say an event is an* elementary *event whenever it represents the outcome of exactly one experiment. We call universe of events (denoted as Ω) the set of all possible elementary events that can occur for a given experiment.*

Definition 1.1.2 (Probability Distribution) *Let Ω be a universe of events and let $\mathcal{F}$ be a σ-algebra on Ω (when Ω is discrete, $\mathcal{F}$ is equal to 2^{Ω}). A* probability distribution $P : \mathcal{F} \mapsto \mathbb{R}$ *is a mapping from set of events $\mathcal{F}$ to the Real Numbers $\mathbb{R}$, such that:*

1. $P(A) \geq 0, \forall A \in \mathcal{F}$,
2. $P(\Omega) = 1$,
3. *If $A, B \in \mathcal{F}$ and $A \cap B = \emptyset$, then $P(A \cup B) = P(A) + P(B)$.*

A probability distribution is said to be positive iff $\forall A \in \mathcal{F}$, such that $A \neq \emptyset$, we have that $P(A) > 0$. From this definition, we can easily prove that $P(\emptyset) = 0$, and that in general $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

We have defined *probability* in terms of (sets of) events. In order to simplify its representation, we often express those events through values (numerical or categorical). To do so, we introduce the concept of random variables (see Definition 1.1.3).

Definition 1.1.3 (Random Variable) *Let Ω be a universe of events, let $\mathcal{F}$ be a σ-algebra on Ω and $(\Omega, \mathcal{F}, P)$ be a probabilistic space. In addition, let Ω_X be a set and let $\mathcal{F}_X$ be σ-algebra on Ω_X. A* random variable $X : \Omega \mapsto \Omega_X$ *is a measurable function such that $\forall x \in \mathcal{F}_X : X^{-1}(x) = \{y \in \Omega : X(y) \in x\} \in \mathcal{F}$. Then, we define the probability over $\mathcal{F}_X$ as $P_X(x) = P(X^{-1}(x))$. We call Ω_X the domain of Variable X. A random variable is said to be categorical if its domain does not represent a quantity, and is said to be discrete whenever its domain has a finite or a countably infinite amount of elements [Starnes et al., 2010]. In the same way, a random variable $\mathring{X}$ is said to be continuous if it is not a discrete one. When it is clear from the context, we write simply $P(\cdot)$ instead of $P_X(\cdot)$.*

For simplicity, hereafter, we will denote the joint probability $P(X_1 \cup \cdots \cup X_n)$ as $P(X_1, \ldots, X_n)$. Also, we will frequently encounter probabilities of the form $P(X_1 = x_1, \ldots, X_n = x_n)$, we will often denote those as $P(x_1, \ldots, x_n)$. At this point, we state that, unless otherwise specified, in the rest of this thesis capital letters $(X, Y, Z, \ldots)$ will always represent random variables, capital bold letters $(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \ldots)$ will represent sets of random variable. Lowercase letters will always represent instances (values) of random variables (or of sets of random variables when in boldface) represented by their corresponding capital letters $(x, y, z, \mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots)$. We state as well that we will use a circle above letters to express that a variable is continuous $(\mathring{X}, \mathring{Y}, \mathring{Z}, \ldots)$, or to mention that a set of random variables contains some continuous variables $(\mathring{\mathbf{X}}, \mathring{\mathbf{Y}}, \mathring{\mathbf{Z}}, \ldots)$.

In Definition 1.1.3 we introduced definitions of discrete and continuous variables which might be too strict to our purposes for the following reasons:

- In a computer context numbers are represented by a fixed number of bits, so any random variable would be inherently discrete.
- Later in this thesis we will use the notion of discrete random variable in a BN context, where having huge domain sizes is prohibitive in terms of inference and learning.

In the context of this thesis, we will consider random variable $\mathring{X}$ to be continuous as long as we can establish an strict order between all its possible outputs, and whenever its domain size is too big to be dealt with a BN. For instance, we will take the liberty to consider $\mathring{X} \in \{0, 1, \ldots, 1000\}$ to be a continuous variable (in other words, we extend $\mathring{X}$ to be defined in interval $[0, 1000]$).

Having introduced the concept of continuous random variables, it is worth noting that the probability of any continuous variable $\mathring{X}$ at any single value $\mathring{x}$ is equal to zero: $P(\mathring{X} = \mathring{x}) = 0$, which can hide the fact that some values of $\mathring{X}$ are more likely to appear than others. To overcome this issue, we measure the probability of a continuous variable belonging to a measurable set of values (e.g. an interval) instead of a single value. We do this by means of probability density functions and cumulative density functions (see Definitions 1.1.4 and 1.1.5).

Definition 1.1.4 (Probability Density Function) *Let $\mathring{X}$ be a random variable of domain $\Omega_{\mathring{X}}$ and let $\mathcal{F}_{\mathring{X}}$ be a σ-algebra on $\Omega_{\mathring{X}}$. A* probability density function *(pdf.) is a non-negative integrable function $f_{\mathring{X}} : \mathcal{F}_{\mathring{X}} \mapsto \mathbb{R}$, such that for any set of events $I \in \mathcal{F}_{\mathring{X}}$:*

$$P_{\mathring{X}}(\mathring{x} \in I) = \int_{\mathring{y} \in I} f_{\mathring{X}}(\mathring{y}) d\mathring{y}$$

Definition 1.1.5 (Cumulative Density Function) *A* cumulative density function *(cdf.) $F_{\mathring{X}}$ is the function that represents the probability of a random variable to be greater than or equal to a given value, and can be defined in terms of a pdf.:*

$$F_{\mathring{X}}(\mathring{x}) = P_{\mathring{X}}(\mathring{X} \leq \mathring{x}) = \int_{-\infty}^{\mathring{x}} f_{\mathring{X}}(\mathring{y}) d\mathring{y}$$

Definition 1.1.6 (Expected Value) *Being X a non-categorical random variable, and P a probability distribution we define its expected value (or mean) as follows:*

$$\mathbb{E}_P[X] = Mean(X) = \sum_{x \in \Omega_X} xP(x)$$

We replace the sum with an integral whenever we deal with continuous variables and pdfs. The subscript P often omitted, when it is clear from the context.

Definition 1.1.7 (Variance, Standard deviation) *Being X a non-categorical random variable (discrete or continuous), we define its variance as:*

$$Var(X) = \mathbb{E}[(X - \bar{X})^2] = \mathbb{E}[X^2] - \bar{X}^2$$

Where $\bar{X} = \mathbb{E}[X]$. We define the standard deviation of X as the square root of its variance:

$$Std(X) = \sqrt{Var(X)}$$

They both measure the spread of numerical values.

When dealing with probabilities in a multivariate context, we are often interested in knowing the probability of a certain configuration of all variables (called *joint* distribution), or also we might be interested in the distribution of a single variable, when all the other variables are unobserved (called a *marginal* distribution). In Definition 1.1.8 we can see those two formally stated, and there is a summation we can perform to find marginal distribution; as we will see in the inference section of this chapter, this method should not be directly applied because that summation might have prohibitive computing time and might even be prohibitive in terms of computer memory when not performed in a smart order.

Definition 1.1.8 (Marginal and Joint Distributions) *For a random variable set $\mathbf{X} = \{X_1, \ldots, X_n\}$, we name $P(X_1, \ldots, X_n)$ a Joint Distribution (or Probability) of $\mathbf{X}$, and we name $P(X_i), \forall i \in \{1, \ldots, n\}$ a Marginal Distribution (or probability) of X_i. These two definitions are related since a Marginal Distribution can be obtained from a Joint Distribution via a* marginalization *process:*

$$\sum_{\mathbf{X} \setminus \{X_i\}} P(X_1, \ldots, X_n) = P(X_i)$$

In the previous equation, whenever we find a continuous variable we replace its corresponding sum by an integral.

We are also interested in measuring how the probability of a variable X can be influenced if another variable Y takes a certain value, which leads to the notion of *conditional probability*:

Definition 1.1.9 (Conditional Probability) *The conditional probability of X given Y is defined as:*

$$P(X|Y) = \frac{P(X, Y)}{P(Y)}$$

Bayes Theorem is a direct consequence of Definition 1.1.9. In Theorem 1.1.1 $P(X)$ is often called an *a priori* and represents a knowledge or belief about X that we would like to take into account for computing the conditional probability $P(X|Y)$. Another consequence of the definition of conditional probabilities is that we can express any joint distribution as a product of conditional probabilities, this is called the Chain Rule (see Theorem 1.1.2).

Theorem 1.1.1 (Bayes Theorem) *Given two random variables X and Y:*

$$P(X \mid Y) = \frac{P(X)P(Y \mid X)}{P(Y)}$$

Theorem 1.1.2 (Chain Rule)

$$P(X_1, \ldots, X_n) = P(X_1)P(X_2 \mid X_1) \ldots P(X_n \mid X_1, \ldots, X_{n-1})$$

Later in this chapter, we will see that BNs (and PGMs in general) do not encode directly conditional dependences, but instead they encore conditional independences (see Definition 1.1.10); moreover, the more independences we find in a phenomenon, the fewer parameters we will need to represent it through a BN.

Definition 1.1.10 (Conditional Independence) *Being $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ random variable sets, and P a probability distribution, we say that $\mathbf{X}$ is conditionally independent of $\mathbf{Y}$ given $\mathbf{Z}$, denoted $P \models (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$, iff: $P(\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Z})$. Also, using the definition of conditional probability we can give an equivalent definition of conditional independence: $P \models (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$, iff $P(\mathbf{X}, \mathbf{Y} \mid \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Z})P(\mathbf{Y} \mid \mathbf{Z})$ In the particular case of $\mathbf{Z} = \emptyset$, we say $\mathbf{X}$ and $\mathbf{Y}$ are marginally independent and we denote this as $P \models (\mathbf{X} \perp\!\!\!\perp \mathbf{Y})$.*

Sets of conditional independences can be represented by a *graphoid*. In [Pearl and Paz, 1986] there is an analysis of how independence assertions can be represented by graph paths (this is why they are called graphoids), filling the gap between statistics and graph theory is certainly the origin of PGMs as such. In the graphoid theory the following five conditional independence properties became axioms:

Definition 1.1.11 (Graphoid axioms)

1. *Symmetry:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{Y} \perp\!\!\!\perp \mathbf{X} \mid \mathbf{Z})$
2. *Decomposition:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$
3. *Weak Union:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}, \mathbf{W})$

4. *Contraction:* $(\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}, \mathbf{Y}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z})$

5. *Intersection (Only for P positive):*
   $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}, \mathbf{W}) \wedge (\mathbf{X} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{Z}, \mathbf{Y}) \implies (\mathbf{X} \perp\!\!\!\perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z})$

If only Axioms 1 to 4 hold, the independence structure is called a semi-graphoid.

1.2 Graphs

In this section we give a brief introduction to graph theory, which will help us to understand the concept of BN in the next sections. We start by reviewing the definition of graph, explaining the notation we use to represent it and mentioning some of its particular cases:

Definition 1.2.1 (Directed/Undirected Graph) *A graph $\mathcal{K} = (\mathbf{X}, \mathcal{E})$ is a data structure formed by a set of nodes*[1] *$\mathbf{X} = \{X_1, \ldots, X_n\}$ and a set of edges $\mathcal{E}$*[2]*, which can be undirected edges: $(X_i - X_j)$, with $i \neq j$; or directed edges: $(X_i \to X_j)$, with $i \neq j$. We consider $(X_i - X_j)$ to be the same undirected edge that $(X_j - X_i)$, in the same way we let $(X_j \leftarrow X_i)$ to denote the directed edge $(X_i \to X_j)$. Finally, among the edges $(X_i \to X_j)$, $(X_j \to X_i)$ and $(X_i - X_j)$, we forbid the existence of more than one in the same graph. A graph $\mathcal{H} = (\mathbf{X}', \mathcal{E}')$ whose set of edges $\mathcal{E}'$ admits only undirected edges is said to be an undirected graph. In the same logic, a graph $\mathcal{G} = (\mathbf{X}'', \mathcal{E}'')$ graph whose set of edges $\mathcal{E}''$ admits only directed edges is said to be a directed graph.*

We denote $(X_i \leftrightarrow X_j)$ to represent the directed edge $(X_i \to X_j)$ or $(X_j \to X_i)$. Then, we let $(X_i \rightleftharpoons X_j)$ represent undirected and directed edges of the forms $(X_i - X_j)$ and $(X_i \leftrightarrow X_j)$, respectively.

In certain situations, we will be interested in analyzing only a portion of a graph, that is why we define also the concepts of subgraph (see Definition 1.2.2). There is a particular case of subgraph that will be especially useful when performing BNs inferences: the cliques (see Definition 1.2.3), we will have a further discussion about cliques in the following sections.

Definition 1.2.2 (Subgraph, Induced Subgraph) *Being $\mathcal{K} = (\mathbf{X}, \mathcal{E})$ a graph, the graph $\mathcal{K}' = (\mathbf{X}', \mathcal{E}')$ is said to be a subgraph of $\mathcal{K}$ iff. $\mathbf{X}' \subset \mathbf{X}$ and $\mathcal{E}' \subset \mathcal{E}$. If $\mathcal{E}'$ is the set all all edges $(X \rightleftharpoons Y) \in \mathcal{E}$ s.t. $X, Y \in \mathbf{X}'$, $\mathcal{K}'$ is said to be an induced subgraph of $\mathcal{K}$ and can be denoted as $\mathcal{K}' = \mathcal{K}[\mathbf{X}']$*

[1] Nodes can be also called *vertices*.

[2] Edges can also be called *arcs*.

Definition 1.2.3 (Clique) *Being $\mathcal{K} = (\mathbf{X}, \mathcal{E})$ a graph, we define a clique as a subset of nodes $\mathbf{C} \subseteq \mathbf{X}$, such that the induced subgraph $\mathcal{K}[\mathbf{C}]$ is complete, i.e. for every couple of nodes $X_i \neq X_j$ in $\mathbf{C}$ there exist an edge of the form $X_i \rightleftharpoons X_j$. We say a clique is a maximal clique of a graph if it is not a subset of another larger clique in the same graph.*

We will be particularly interested in analyzing paths and trails in graphs (see Definition 1.2.4), since their presence (or absence) will determinate probabilistic independences in a graph whenever we let the nodes act as random variables (see Definition 1.1.11, also more on this in the following sections). In PGMs in general, graph's edges represent interaction between random variables, this implies that if all variables are interacting, the graph that represents them will be connected (see Definition 1.2.5).

Definition 1.2.4 (Trail, Path) *Being a graph $\mathcal{K} = (\mathbf{X}, \mathcal{E})$, the sequence $(X_1, \ldots, X_k)$ is a trail whenever $\forall i = \{1, \ldots, k-1\} : (X_i \rightleftharpoons X_{i+1}) \in \mathcal{E}$, and all its elements $\{X_1, \ldots, X_k\}$ are distinct (except possibly the first and the last ones). The trail $(X_1, \ldots, X_k)$ is a path in $\mathcal{K}$ if $(X_i \leftarrow X_{i+1})$ does not belongs to $\mathcal{E}, \forall i = \{1, \ldots k-1\}$. If there is at least one directed edge forming the path, we say it is a directed path.*

Definition 1.2.5 (Connected Graph, Tree) *A graph $\mathcal{K} = (\mathbf{X}, \mathcal{E})$ is a connected graph whenever for any two different nodes $X_i, X_j \in \mathbf{X}$ there exists a trail $(X_i, \ldots, X_j)$. An undirected graph $\mathcal{H} = (\mathbf{X}, \mathcal{E})$ is a Tree whenever for any two different nodes $X_i, X_j \in \mathbf{X}$ a path $(X_i, \ldots, X_j)$ exists and is unique.*

For explaining PGMs properties we need to be capable to describe the basic relations between nodes within a graph. In a directed graph we will talk about ancestors, descendants, parents, children and Markov blanket of a node (see Definitions 1.2.6, 1.2.7 and 1.2.9) whereas for an undirected graph we will talk about neighbors (see Definition 1.2.8).

Definition 1.2.6 (Ancestors, Descendants) *Being $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ a directed graph, for two different nodes $X_i, X_j \in \mathbf{X}$, X_i is an ancestor of X_j if there exists a directed path $(X_i, \ldots, X_j)$. X_j is a descendant de X_j whenever X_i is an ancestor of X_j.*

Definition 1.2.7 (Parents, Children) *Being $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ a directed graph, we define the parents of X_i in $\mathcal{G}$, $\mathbf{Pa}_{\mathcal{G}}(X_i)$, as the set of nodes such that $X_j \in \mathbf{Pa}_{\mathcal{G}}(X_i)$ iff $(X_j \rightarrow X_i) \in \mathcal{E}$. In the same way, we define the children of X_i, $\mathbf{Ch}_{\mathcal{G}}(X_i)$ as the set of nodes such that $X_j \in \mathbf{Ch}_{\mathcal{G}}(X_i)$ iff $(X_i \rightarrow X_j) \in \mathcal{E}$.*

Definition 1.2.8 (Neighbors) *Being $\mathcal{K} = (\mathbf{X}, \mathcal{E})$ a graph, having $X \in \mathbf{X}$, we denote the neighbors of X in $\mathcal{K}$ as $\mathbf{Ngb}_{\mathcal{K}}(X)$ and define it as $Y \in \mathbf{Ngb}_{\mathcal{K}}(X)$ iff there exists an edge $(X - Y) \in \mathcal{E}$.*

Definition 1.2.9 (Markov Blanket) *[Pearl, 1988] Being $\mathcal{G}$ a directed graph over the node set $\mathbf{X}$, having $X \in \mathbf{X}$, we denote by $\mathbf{MB}_{\mathcal{G}}(X)$ the Markov blanket of X and we define it as the union of the parents, the children and other children's parents of X:*

$$\mathbf{MB}_{\mathcal{G}}(X) = \mathbf{Pa}_{\mathcal{G}}(X) \cup \mathbf{Ch}_{\mathcal{G}}(X) \cup \left[\bigcup_{Y \in \mathbf{Ch}_{\mathcal{G}}(Y)} \mathbf{Pa}_{\mathcal{G}}(ch) \right]$$

Some PGMs, in particular BNs, require directed graphs having no cycles (see Definition 1.2.10). Those graphs are called directed acyclic graphs (DAGs, see Definition 1.2.11). Within DAGs, we can always define a topological order of the nodes (see Definition 1.2.12). This is important in our context because it will be the order in which we can perform exact random sampling in a BN, and also because it is an order we can impose to certain BN learning methods (this will be discussed in the following sections).

Definition 1.2.10 (Cycle) *A cycle in a graph $\mathcal{K}$ is defined as the sequence $(X_1, \ldots, X_n)$ if:*

1. *$(X_1, \ldots X_n, X_1)$ is directed path in $\mathcal{K}$, whenever the graph is directed.*
2. *$(X_1, \ldots X_n, X_1)$ is a trail in $\mathcal{K}$, whenever the graph is undirected.*

Definition 1.2.11 (Directed Acyclic Graph) *A directed graph graph (DAG) is a directed graph which has no cycles.*

Definition 1.2.12 (Tolological Order) *Being $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ a directed graph, the sequence $(X_1, \ldots, X_n)$ is a topological order of $\mathcal{G}$ whenever $\forall (X_i \to X_j) \in \mathcal{E}, i < j$. It can be easily shown that a topological order exists iff $\mathcal{G}$ is a DAG.*

1.3 Bayesian Networks Definition

Having discussed about the statistics foundations and about graph theory, in this section we introduce the remaining concepts to give a formal definition of a BN. PGMs encode sets of independences (see Definition 1.3.1), which follow the *semi-graphoid* or the *graphoid* axioms (see 1.1.11). In this context a BN structure is a DAG which respects the local Markov property: each variable is independent of any of its non-descendants when their parents' values are observed (see Definition 1.3.2).

Definition 1.3.1 (Set of Independences) *The independence set $\mathcal{I}(P)$ is the set of assertions $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ that hold in the probabilistic distribution P.*

Definition 1.3.2 (Bayesian Network Structure) *A BN Structure $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ is a DAG whose nodes $\mathbf{X} = \{X_1, \ldots, X_n\}$ represent random variables and in which the following set of conditional local independences are encoded:*

$$\forall X_i \in \mathbf{X} : (X_i \perp\!\!\!\perp \mathbf{NonDesc}(X_i) \mid \mathbf{Pa}(X_i))$$

Where $\mathbf{NonDesc}(X_i)$ and $\mathbf{Pa}(X_i)$ are the sets of non-descendants and parents of X_i in $\mathcal{G}$, respectively.

In BN structures, the probabilistic dependency between random variables (their nodes) is determined by active trails, which are defined in the context of a subset of observed variables (see Definition 1.3.3). If there is no active trail between two variables, we consider them as d-separated (see Definition 1.3.4).

Definition 1.3.3 (Active Trail) *[Geiger and Pearl, 1990] Being $\mathcal{G}$ a BN structure, and $(X_1, \ldots, X_k)$ a trail in $\mathcal{G}$, we consider it to be an active trail given a set of observed variables $\mathbf{Z}$ if, for all* v-structures[3] *$X_{i-1} \rightarrow X_i \leftarrow X_{i+1}\, (1 < i < k)$ in $\mathcal{G}$, X_i* or one of its descendants is in $\mathbf{Z}$ *, and in no other case there is a X_i in that trail belonging to $\mathbf{Z}$.*

Definition 1.3.4 (d-separation) *[Pearl, 1988]. Being $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ three node sets belonging to the BN structure $\mathcal{G}$, we define $\mathbf{X}$ and $\mathbf{Y}$ to be d-separated by $\mathbf{Z}$, if there does not exist any active trail between any node $X \in \mathbf{X}$ to a node $X \in \mathbf{Y}$ given $\mathbf{Z}$ over $\mathcal{G}$. This is denoted d-sep$_{\mathcal{G}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$. Also, we denote $\mathcal{I}(\mathcal{G})$ the set of independences corresponding to d-separations in the following way:*

$$\mathcal{I}(\mathcal{G}) = \{(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z}) : \textit{d-sep}_{\mathcal{G}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})\}$$

Theorem 1.3.1 *[Pearl, 1988] In a BN structure $\mathcal{G}$ any variable X_i is d-separated by its Markov blanket,* i.e.*:*

$$(X_i \perp\!\!\!\perp X_j | \mathbf{MB}_{\mathcal{G}}(X_i)) \in \mathcal{I}(\mathcal{G})$$

Where $j \neq i$ and $X_j \notin \mathbf{MB}_{\mathcal{G}}(X_i)$.

There are two key concepts defined over BN structures: first, *I-map* (see Definition 1.3.5), which expresses whether the independences encoded in a BN structure are a subset of another set of independences (for instance, the ones encoded

[3]The form $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ is said to be a *v-structure*.

in a probabilistic distribution); second, *factorization* (see Definition 1.3.6), which defines an algebraic way to express a probabilistic distribution related to a BN structure. It is showed in Theorems 1.3.2 and 1.3.3 that both concepts are equivalent, and moreover in Theorem 1.1.2 it is shown that for any possible DAG, there is always a probabilistic distribution that factorizes over it.

Definition 1.3.5 (I-map, Minimal I-map) *Being $\mathcal{G}$ a BN structure, which holds the set of independences $\mathcal{I}(\mathcal{G})$, we call $\mathcal{G}$ an independence map (I-Map) of a set of independences $\mathcal{I}$ whether $\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}$. In the same way we can say that $\mathcal{G}$ is an I-map of a probability distribution P whether $\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}(P)$. $\mathcal{G}$ is said to be a* minimal *I-map of $\mathcal{I}$ whenever removing any edge from $\mathcal{G}$ would result in it being no longer an I-map.*

Definition 1.3.6 (Factorization) *Being $\mathcal{G}$ a BN structure over the random variable set $\mathbf{X}$, we consider $\mathcal{G}$ to factorize a probability distribution P whenever P can be expressed as:*

$$P(\mathbf{X}) = \prod_{X_i \in \mathbf{X}} P(X_i \mid \mathbf{Pa}_{\mathcal{G}}(X_i))$$

Theorem 1.3.2 *[Geiger and Pearl, 1990] Being $\mathcal{G}$ a BN structure over random variables set $\mathbf{X}$, and P a probability distribution over $\mathbf{X}$, if $\mathcal{G}$ is an I-map of P, then $\mathcal{G}$ factorizes P.*

Theorem 1.3.3 (d-separation Soundness) *[Geiger and Pearl, 1990] Being $\mathcal{G}$ a BN structure and P a probabilistic distribution. If $\mathcal{G}$ factorizes P, then $\mathcal{G}$ is an I-map of P.*

Theorem 1.3.4 (d-separation Completeness) *[Geiger and Pearl, 1990] Being $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ a BN structure and P a probabilistic distribution: for every $X, Y \in \mathbf{X}$ and $\mathbf{Z} \subseteq \mathbf{X}$.*

$$(X \perp\!\!\!\perp Y \mid \mathbf{Z}) \notin \mathcal{I}(\mathcal{G}) \implies \exists P : (X \perp\!\!\!\perp Y \mid \mathbf{Z}) \notin \mathcal{I}(P) \text{ s.t. } \mathcal{G} \text{ factorizes } P$$

Since for any possible DAG there will be a distribution that factorizes into it, we can define specific parameters for each factor of that distribution. Here we present one way to assign those parameters: the case of discrete BNs (see Definition 1.3.7), where we deal with discrete finite-domain variables, and we store all probabilistic distribution into conditional probabilities tables (CPTs).

Definition 1.3.7 (Bayesian Network) *[Pearl, 1988] A (discrete) BN $\mathcal{B}$ is a pair $(\mathcal{G}, \boldsymbol{\theta})$ where:*

- $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ *is a BN structure*
- $\mathbf{X} = \{X_1, \ldots, X_n\}$ *represents a set of discrete random variables (by abuse of notation, it is usual to use interchangeably* $X_i \in \mathbf{X}$ *to denote a node in the BN and its corresponding random variable),*
- $\mathcal{E}$ *is a set of edges (arcs),*
- $\boldsymbol{\theta} = \{P(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i))\}_{i=1}^{n}$ *is the set of the conditional probability tables (CPTs) of the variables* X_i *in* $\mathcal{G}$ *given their parents* $\mathbf{Pa}_{\mathcal{G}}(X_i)$.

The BN factorizes the joint probability P *over* $\mathbf{X}$ *as follows:*

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i)). \tag{1.1}$$

As an example, Figure 1.1 displays a BN over random variables $\mathbf{X} = \{A, B, C, D, E\}$. Here, the joint probability over $\mathbf{X}$ is:

$$P(A, B, C, D, E, F) = P(A)P(B|A)P(C)P(D|C)P(E|B, D)P(F|E)$$

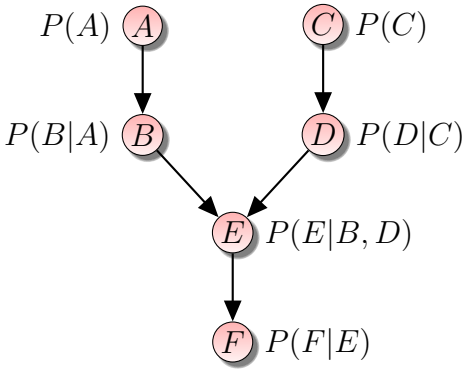


Figure 1.1: A Bayesian network.

In order to factorize P it is convenient to use a graph $\mathcal{G}$ which is a minimal I-map of P, because it results in fewer parameters when constructing a BN (which is one of the main goals of BNs and of PGMs in general: exploit independences to construct compact multivariate representations). We can add to the problem of representing P a D-map perspective (see Definition 1.3.8): if $\mathcal{G}$ is both, a D-map and a I-map of P, it would represent exactly the same independences of P, *i.e.* it would be a P-map of P, which is the best possible representation of P. Unfortunately, P-maps do not always exist for two reasons: first, P might

encode deterministic relations between variables and second, BN independence assumptions are not appropriate for every case. However, the conditions of a distribution P to have a P-map are presented in Theorem 1.3.5, which does not allow zero-measurements on the CPDs representations of P, implying that P must be a strictly positive distribution, hence *a fortiori*, that it encodes no deterministic relation between its variables.

Definition 1.3.8 (D-map, Maximal D-map) *We say a BN structure $\mathcal{G}$ is an D-map of a distribution P if $\mathcal{I}(\mathcal{G}) \supseteq \mathcal{I}(P)$. We say $\mathcal{G}$ is a maximal D-map of P if adding any edge into it results in it being no longer a D-map.*

Definition 1.3.9 (P-map, faithfulness) *We say a BN structure $\mathcal{G}$ is a perfect map (P-map) of a distribution P whenever $\mathcal{G}$ is, at the same time, an I-map and a D-map of P. In that case we also say that $\mathcal{G}$ is faithful to P,* i.e. $\mathcal{I}(P) \equiv \mathcal{I}(G)$.

Theorem 1.3.5 *[Meek, 1995b] For almost all distributions P [4] that factorize over $\mathcal{G}$, we have that $\mathcal{G}$ is a P-map of P.*

If a P-map exists for a distribution P, it is not unique because different graphs can encode exactly the same set of independences. We call this case Markov equivalence (see Definition 1.3.10). The Markov equivalence between two DAGs is ruled by Theorem 1.3.6.

Definition 1.3.10 (Markov equivalence class) *We call a Markov equivalence class the set of graphs $\{\mathcal{G}_1, \ldots, \mathcal{G}_p\}$ such that all their elements encode the same independences set: $\mathcal{I}(\mathcal{G}_1) = \ldots = \mathcal{I}(\mathcal{G}_p)$. Any pair of graphs in that set are said to be Markov equivalent.*

Definition 1.3.11 (Immorality) *The v-structure $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ in a graph $\mathcal{G}$ is called immorality whenever the is no edge between X_{i-1} and X_{i+1}.*

Theorem 1.3.6 (Markov Equivalence) *[Verma and Pearl, 1991] Two DAGs are Markov equivalent iff they have the same skeleton [5], and the same immoralities.*

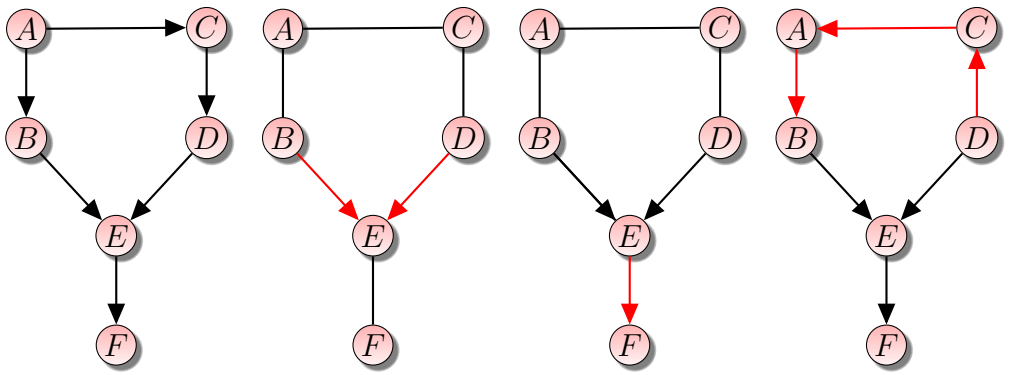
We can express Markov equivalence classes in a graphical way using partially oriented DAGs, which we call CPDAGs (see Definition 1.3.12). In order to build a CPDAG, we must take the skeleton of a DAG and orient the immorality edges in their original direction; some undirected edges might have only one possible direction in order to avoid creating new immoralities, we finally replace those with directed edges in their original directions. This concept is applied in Example 1.3.1.

[4] In this context, *almost all* refers to all distributions except those with a zero-measure set in the conditional probability distribution space.

[5] *Skeleton* is the undirected graph that results from replacing every directed edge with an undirected one.

Definition 1.3.12 (CPDAG) *(Completed partially-oriented DAG) A CPDAG of a DAG $\mathcal{G}$ is the (unique) graph which is the result of replacing all directed edges of $\mathcal{G}$ with undirected whenever they do not make part of any immorality and only if reverting their direction would not generate a new immorality.*

Example 1.3.1 (Building a CPDAG) *In Figure 1.2a, we have a DAG $\mathcal{G}$ whose (unique) CPDAG we intend to find. The first step to do so is to capture its skeleton and orient its immorality edges (in this case, only $B \rightarrow E \leftarrow D$), which is done in Figure 1.2b (see red edges), giving a partially oriented DAG (PDAG). We notice that in a Markov equivalence class the edge between E and F must be oriented as $E \rightarrow F$, because otherwise it would generate new immoralities around E; the rest of undirected arcs are not compelled into any direction. Then the resulting CPDAG of $\mathcal{G}$ is presented in Figure 1.2c. The CPDAG can be used to easily instantiate new graphs belonging to the same Markov equivalence class of $\mathcal{G}$, in order to do so, we must orient the undirected edges in any possible way such that it does not generate any new immoralities, e.g the graph presented in Figure 1.2d.*



(a) A graph $\mathcal{G}$. (b) PDAG of $\mathcal{G}$. (c) CPDAG of $\mathcal{G}$. (d) Equivalent of $\mathcal{G}$.

Figure 1.2: Creating a CPDAG from $\mathcal{G}$.

1.4 Inference

In this section we discuss about inference methods in a BN context. We call inference any process of solving a probability query. There are three main types of probability query in the BN context: First, the conditional probability query, which consists in the computation of the probability of a set of variables, possibly having evidence (see Definition 1.4.1); second, the Maximum a Posteriori (MAP) inference, in which we look for the instances of a set of variables that maximize a

given conditional probability query [Pearl, 1988]. As a special case, when the set of variables contains all the variables in the BN, we get a Most Probable Explanation (MPE) query. Third, the Most Relevant Explanation (MRE) inference, in which we look for a relevant subset of variables to explain a given evidence [Yuan and Lu, 2008].

Definition 1.4.1 (Conditional Probability Query) *Being $P(\mathbf{X})$ a probability distribution defined over the random variable set $\mathbf{X}$, having a query variable set $\mathbf{Q} \subset \mathbf{X}$, and an evidence variable set $\mathbf{E} \subset \mathbf{X}$, a conditional probability query is defined as the following computation:*

$$P(\mathbf{Q}|\mathbf{E} = \mathbf{e}) = \frac{\sum_{\mathbf{Y}} P(\mathbf{Y}, \mathbf{Q}, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})},$$

where $\mathbf{Y} = \mathbf{X} \backslash (\mathbf{Q} \cup \mathbf{E})$.

Inference over conditional probabilities queries is not a trivial task, it is proven in [Cooper, 1990] that it happens to be a NP-Hard problem; actually even approximate inference is proven to be a NP-Hard problem [Dagum and Luby, 1993]. Unsurprisingly, MPE inference turns out also to be a NP-hard problem, which is proven in [Shimony, 1994]. However, MAP inference is in general a harder problem (NP^{pp}-complete) but it reduces to a NP-complete problem if the BN is a polytree [Park and Darwiche, 2004]. MRE inference is also a complex: it is known to be in NP^{pp} and it is conjectured to be NP^{pp}-complete [Yuan et al., 2011].

There are four main kinds of methods for inference of type conditional probability inference. First, classic exact inference methods such as the Variable elimination (VE) algorithm [Dechter, 1999], and Clique-Tree message passing algorithms such as Shafer-Shenoy [Shenoy and Shafer, 1990] and Lazy Propagation [Madsen and Jensen, 1999]. Second, there are methods that treat inference as an optimization problem, the most notable among them being the "Loopy" Belief Propagation (which is an approximate method) and its improved version: the Generalized Belief Propagation [Yedidia et al., 2001]. Third, sampling-based methods, like logic sampling [Henrion, 1988] or Gibbs Sampling [Geman and Geman, 1984], which provide approximate solutions but are very useful whenever exact inference methods are infeasible. Fourth, methods based in weighted-model-counting in which BNs are encoded into Conjunctive Normal Form models, each of which has an assigned weight, allowing the representation of a probability query exactly as the sum of weights of the models that are consistent given some evidence [Chavira and Darwiche, 2006, Chavira and Darwiche, 2008]. MPE inferences can also be

solved using some Clique-Tree message passing algorithms but using constrained trees (resulting from constrained elimination orders) [Mateescu et al., 2010], or by branch and bound-like techniques [Park and Darwiche, 2003]. Finally, MRE can be solved using MCMC-based algorithms [Yuan and Lu, 2008]. In this thesis, we focus on solving conditional probability queries on BNs.

In the rest of this section we discuss about the VE and the Shafer-Shenoy algorithms, which are both exact methods and which are both equivalent in terms of computation times, as we will see later. Those will be the only inference methods addressed in the present thesis.

1.4.1 Variable Elimination

We will introduce the VE algorithm by means of an example:

Example 1.4.1 (Variable elimination by hand) *We want to compute the probability of $P(F)$ in the BN represented in Figure 1.1, having no evidence variables for this particular query.*
Since $P(F) = \sum_{\mathbf{X}\setminus\{F\}} P(\mathbf{X}) = \sum_A \dots \sum_E P(A, \dots, F)$, we can perform these summations (variable eliminations) in many different orders, obtaining always the same result. However, we will see later in this section that elimination orders can have a big impact on the time required to perform this computation as well as in the memory required to store all its intermediate results. For this example we choose to perform the computations in the order shown in Equation 1.2.

$$P(F) = \sum_E \left(\sum_B \left(\sum_D \left(\sum_C \left(\sum_A P(A,B,C,D,E,F) \right) \right) \right) \right) \tag{1.2}$$

$$P(B,C,D,E,F) = \underbrace{\left(\sum_A P(A)P(B|A) \right)}_{P(B)} P(C)P(D|C)P(E|B,D)P(F|E) \tag{1.3}$$

$$P(B,D,E,F) = P(B) \underbrace{\left(\sum_C P(C)P(D|C) \right)}_{P(D)} P(E|B,D)P(F|E) \tag{1.4}$$

$$P(B,E,F) = P(B)\underbrace{\left(\sum_{D} P(D)P(E|B,D)\right)}_{P(E|B)}P(F|E) \tag{1.5}$$

$$P(E,F) = \underbrace{\left(\sum_{B} P(B)P(E|B)\right)}_{P(E)}P(F|E) \tag{1.6}$$

$$P(F) = \sum_{E} P(E)P(F|E) \tag{1.7}$$

The elimination of A and C (see Equations 1.3 and 1.4) generated two CPTs to be stored as intermediate computations: $P(B)$ and $P(D)$. The elimination of D (see Equation 1.4) brings us an intermediate calculation of $P(E|B)$ which is also a CPT: in the original BN, there is no active trail from B to D when no variable is observed, so that $(B \perp\!\!\!\perp D)$, allowing us to assert that $P(D) = P(D|B)$ which leads to the conclusion that: $P(D) \times P(E|B,D) = P(D|B) \times P(E|B,D) = P(E,D|B)$, hence, after summing over D, we get $P(E|B)$. In fact, it is proven in [Butz et al., 2010] that the results of all intermediate computations, even when there are evidence, are always conditional probability distributions. But this information turns out to be irrelevant for our final results, i.e. we do not need to realize to which CPTs correspond the intermediate computations. Hence, in the computations, $P(E|B)$ can simply be interpreted as a function (we call it a potential *or a* factor*) $\phi(B,E)$. It is actually convenient for us to have a structure more flexible than CPTs to store those computations. Moreover, we might want to do inference computations in an unnormalized manner, which could help to gain numerical accuracy in certain cases, and which could also allow us to introduce* beliefs *as computation nodes, which are not probabilities.*
We finish the example eliminating B and E (see Equations 1.6 and 1.7), denoting all the probabilities we had been computing as factors without actually worrying about CPTs generated during the process.

To cope with the need of the computations of Equations 1.5, 1.6 and 1.7 we need to introduce the concept of Factors (see Definitions 1.4.2 and 1.4.3), which are a generalization of CPTs, over which we formally define the operations of Product and Marginalization and Normalization (see Definitions 1.4.4 and 1.4.5). In order to eventually use factors as joint distribution, normalization is defined over factors (see Definition 1.4.6).

Definition 1.4.2 (Factor) *Being $\mathbf{X}$ a random variable set, we define a factor ϕ* [6] *as a function $\phi : \Omega_{\mathbf{X}} \mapsto \mathbb{R}_{\geq 0}$, from the domain of $\mathbf{X}$ to the non-negative real*

[6] Factors are also called Potentials.

numbers. In this context, we also define $\mathbf{X}$ *to be the scope of the factor and we denote it* $\mathbf{X} = Scope(\phi)$.

Definition 1.4.3 (Factor Scope and Size) *Being* $\phi(\mathbf{X})$ *a factor over the random variable set* $\mathbf{X}$*, we define:*

1. *The scope of the factor* ϕ *as* $\mathbf{X} = Scope(\phi)$.
2. *The size of the factor* ϕ *as* $\prod_{X \in \mathbf{X}} |\Omega_X|$.

In addition, if Φ *is a set of factors, we define* $Scope(\Phi) = \bigcup_{\phi \in \Phi} Scope(\phi)$.

Definition 1.4.4 (Factor Product) *Being* $\mathbf{X}, \mathbf{Y}$ *and* $\mathbf{Z}$ *three disjoint random variables sets, we let* $\phi_1(\mathbf{X}, \mathbf{Y})$ *and* $\phi_2(\mathbf{Y}, \mathbf{Z})$ *be two factors. We define the factor product* $\phi_1 \times \phi_2$ *as:*

$$\phi_{prod}(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z}) = \phi_1(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \times \phi_2(\mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z})$$

Definition 1.4.5 (Factor Marginalization) *Being* $\mathbf{X}$ *a random variable set, and letting* $\phi(\mathbf{X})$ *be a factor, the marginalization of* ϕ *w.r.t.* $X \in \mathbf{X}$ *is another factor* $\phi_{marg} : \mathbf{X}_{marg} \mapsto \mathbb{R}_{\geq 0}$*, where* $Scope(\phi_{marg}) = \mathbf{X}_{marg} = (\mathbf{X} \setminus \{X\})$*, such that:*

$$\phi_{marg}(\mathbf{X}_{marg}) = \sum_{x \in \Omega_X} \phi(\mathbf{X}_{marg}, X = x)$$

Definition 1.4.6 (Factor Normalization) *Being* $\phi(\mathbf{X})$ *a factor defined over the random variable set* $\mathbf{X}$*, we define its normalization as a probability distribution* $P_\phi(\mathbf{X})$ *such that:*

$$P_\phi(\mathbf{X}) = \frac{\phi(\mathbf{X})}{\sum_{\mathbf{x} \in \Omega_\mathbf{X}} \phi(\mathbf{X} = \mathbf{x})}$$

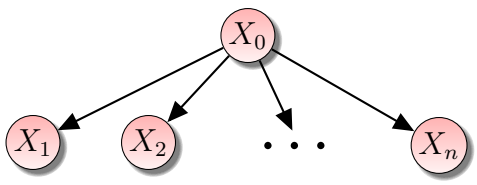


Figure 1.3: A Naive-Bayes Bayesian network with $n + 1$ nodes.

Example 1.4.2 (A Bad Elimination Order) *In Figure 1.3 we present the Naive Bayes network, a particular case of BN which is typically used for classification (having X_0 as class variable). For the purposes of this example, we decide to perform a VE over the naive BN in order to compute $P(X_n)$. To do so, we (badly) choose to use the elimination order $(X_0, X_1, \ldots, X_{n-1})$:*

$$P(\mathbf{X}) = P(X_0)P(X_1|X_0)\ldots P(X_n|X_0) \tag{1.8}$$

$$P(X_n) = \sum_{X_{n-1}} \left(\sum_{X_{n-2}} \ldots \underbrace{\left(\sum_{X_0} P(X_0)P(X_1|X_0)\ldots P(X_n|X_0) \right)}_{\phi(X_1,\ldots,X_n)} \ldots \right) \tag{1.9}$$

We appreciate that something horrible happens in Equation 1.9: the factor $\phi(X_1, \ldots, X_n)$. To understand why is this so bad, let's imagine each of the random variables is binary and let's consider that each element of its CPTs is encoded with double precision, we can assume each number can occupy 8 bytes in memory. For $n = 100$, the CPTs need $2 + 4n \times 8bytes \approx 3.2KB$ in memory to be correctly allocated, whereas the factor $\phi(X_1, \ldots, X_n)$ would require $2^n \times 8bytes \approx 10^{19}TB$, which is completely prohibitive in terms of memory, and in terms of computation time for marginalizing over that factor. This observation is generalized in [Cooper, 1990], where it is shown that the inference time (and its memory requirement) grows exponentially with the size of the largest factor scope generated during the inference process. In consequence, we must look for the elimination order that minimizes that scope size. Unluckily, finding the optimal VE turns out to be a NP-complete problem [Arnborg et al., 1987] (apart from the exact inference problem to solve, which is already NP-hard itself). In practice the elimination order is chosen using heuristic criteria, which will be discussed in the following section.

When we are dealing with inferences of the form $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$, having the evidence variable set $\mathbf{E}$, it is often convenient to express the information of each random variable as a separate random variable itself: it allows us to treat that information as an additional node in the BN and it allows us to express uncertain information about those evidence variables. Therefore, evidence is defined as a random variable conditioned by its respective evidence variable (see Definition 1.4.7).

Definition 1.4.7 (Evidence) *Being X a random variable, we define an evidence e_X as any information we have on the values that occur for X. Such information is entered into the BN as a likelihood measure $P(e_X|X)$.*

Example 1.4.3 (Variable Elimination with Evidence) *We want to compute $P(F|e_B)$ from the BN shown in Figure 1.4, which is the same as the one used for Example 1.4.1, but adding evidence for Variable B. We could compute this as $P(F|e_B) = \sum_{\mathbf{X}\setminus\{F\}} P(\mathbf{X}|e_B) = \sum_A \ldots \sum_E P(A, \ldots, F|e_B)$, but in order to take advantage of the evidence representation we prefer to compute it as follows:*

$$\phi(F, e_B) = \sum_A \ldots \sum_E \phi(A, B, C, D, E, F, e_B) \tag{1.10}$$

$$\phi(F, e_B) = \sum_A \ldots \sum_E P(A) \underbrace{P(B|A)P(e_B|B)}_{\phi(A,B,e_B)} P(C)P(D|C)P(E|B, D)P(F|E) \tag{1.11}$$

$$P(F|e_B) \propto \phi(F, e_B) \tag{1.12}$$

In Equation 1.10, $\phi(A, \ldots, F, e_B)$ does not necessarily represent a joint probability, since $P(e_B|B)$ can represent beliefs w.r.t. the random variable B which are not necessarily a conditional probability. As a consequence, $\phi(F, e_B)$ is proportional (and not necessarily equal) to the marginal distribution $P(F|e_B)$. In Equation 1.11 we let $\phi(A, B, e_B)$ be the product between the CPT $P(B|A)$ and the evidence factor $P(e_B|B)$, which allows us to perform the computations in the same way as in Example 1.4.1. In order to obtain the desired distribution, we apply a normalization procedure according to Definition 1.4.6 in Equation 1.12.

The procedure of the preceding example is generalized in the Variable Elimination Algorithm [Dechter, 1999], which takes as input a subset of query variables $\mathbf{Q}$ (those whose probability will be computed), all the factors of the BNs (CPTs), and all the evidence nodes' factors; giving as output the probability $P(\mathbf{Q})$ (see Algorithm 1.1).

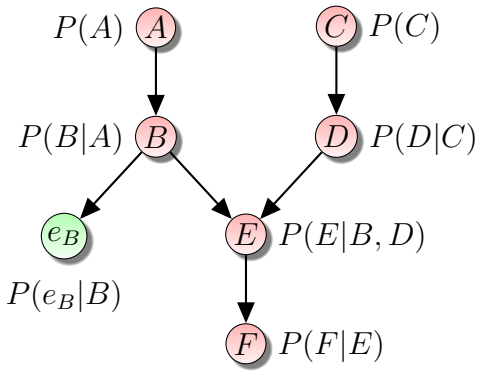


Figure 1.4: A Bayesian network representation with an evidence node for Variable B.

Algorithm 1.1: Variable elimination algorithm.

Input: Φ: a factor set, $\mathbf{Q}$: a query variables set, $\mathbf{e}$: an evidence set, t: a VE order

Output: $\phi_{\mathbf{Q}}$: a factor over $\mathbf{Q}$ encoding $P(\mathbf{Q}|\mathbf{e})$

```
for i ← 1, ..., size(t) do
    X ← i-th element of t;
    if X ∉ Q then
        Φ_X ← set of factors of all φ_X ∈ Φ such that X ∈ Scope(φ_X);
        φ   ← product of all factors φ_X ∈ Φ_X;
        foreach e ∈ e s.t. e is an evidence on X do
            φ ← φ × P(e|X);
        marginalize X from φ;
        remove Φ_X's elements from Φ;
        insert φ into Φ;
// Notice that Φ became the set of factors
// generated during the VE procedure
φ_Q ← product of all factors of Φ;
normalize φ_Q;
return φ_Q;
```

It often happens that not all variables in a BN have influence in the result of an inference. If a variable is unobserved and it has no children in a BN and is not part of our query variables, we can remove it without changing the result of the inference. The systematic process of removing such nodes and their adjacent edges was proposed in [Shachter, 1986], which consists in the elimination of the

so-called *Barren nodes* (see Definition 1.4.8).

Definition 1.4.8 (Barren nodes) *[Shachter, 1986] In the context of an inference process, a node of a BN is considered to be a Barren node whenever it does not belong to the query node set, it has no evidence and if it has children, they are all Barren nodes as well.*

Example 1.4.4 (Barren Nodes Elimination) *In the naive BN from Example 1.4.2, since no variable is observed, the variables $X_1, \ldots, X_{n-1}$ would be barren nodes (as they have no children). So we could have performed VE over a BN with a single edge: $X_0 \rightarrow X_n$, giving as result: $P(X_n) = \sum_{X_0} P(X_0)P(X_n|X_0)$.*

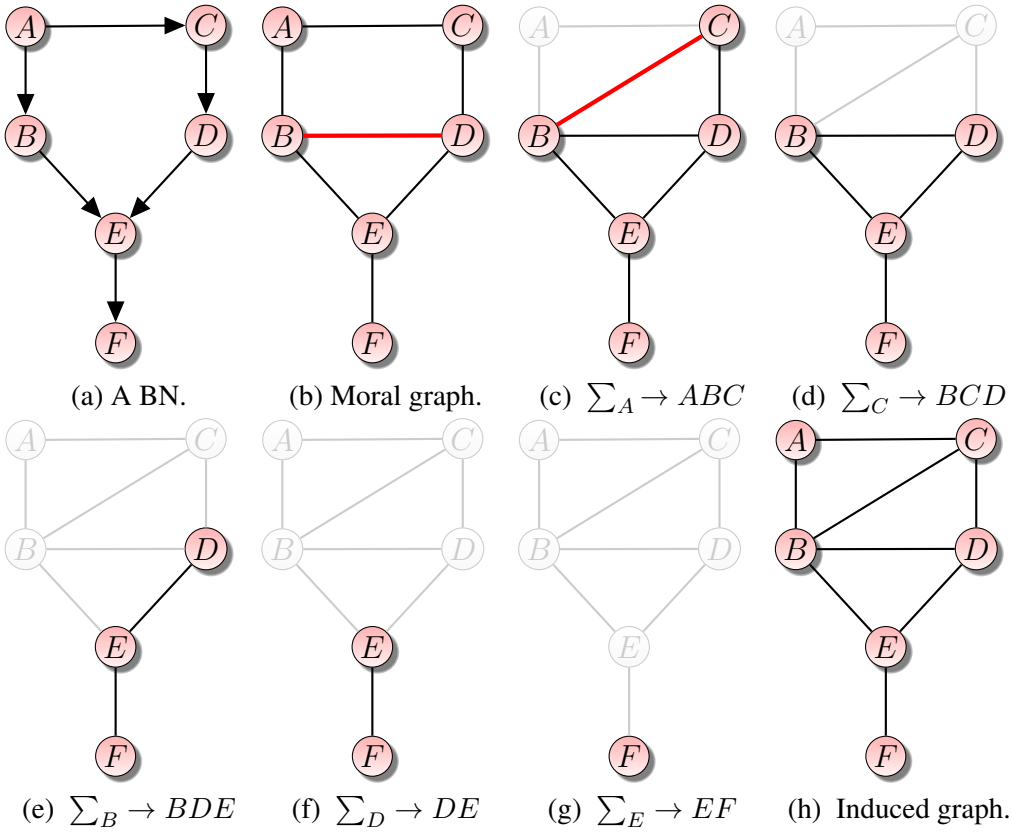
1.4.2 Junction Tree Inference

In the preceding section we dealt with the process of inference using the VE algorithm, expressing CPTs of a BN and their evidence node into factors $\phi(\cdot)$, which do not contain information about the direction of the edges in the BN structure. Therefore, we can represent the factors generated during VE in an undirected graph. We call it an Induced graph (see Definition 1.4.9); we explain how it is created in the following example:

Definition 1.4.9 (Induced graph) *Being Φ a set of factors generated during a VE procedure, its induced graph is the undirected graph $\mathcal{K} = (\mathbf{X}, \mathcal{E})$, where $\mathbf{X} = Scope(\Phi)$ and $(X - Y) \in \mathcal{E}$ iff $\exists \phi \in \Phi$ such that $\{X, Y\} \subseteq Scope(\phi)$.*

Example 1.4.5 (Creation of an induced graph) *We perform a VE process to compute $P(F)$ in the BN shown in Figure 1.5a. The first step is to create an undirected graph having the same nodes of the BN, with the directed edges of the BN converted into undirected ones (i.e., this is the skeleton of the BN). The next step is to add to this graph an undirected edge between every pair of nodes sharing a common child in the original BN (if such an edge does not already exists): since all the CPTs are of the form $P(X|Y_0, \ldots, Y_j)$, they have associated factors of the form $\phi(X, Y_0, \ldots, Y_j)$ and the newly added edges account for these factors. The result of this step is called the moral graph (see Definition 1.4.10), in our example the result is shown in Figure 1.5b. Then we proceed the elimination using Order A, C, B, D, E. In the elimination of Variable A, we compute the following marginalization $\sum_A P(A)P(B|A)P(C|A)$, for which we have to generate $\phi(A, B, C)$, which we call simply factor ABC; this implies adding the edge $B - C$ for building the desired induced graph (see Figure 1.5c). In the same logic, we eliminate A, C, B, D and E, generating the factors BCD, BDE, DE and EF respectively. (see Figures 1.5c,1.5d,1.5e, and 1.5f). In general when we eliminate a variable, we remove its adjacent edges and we add edges between all its*

neighbors, creating a clique; it is proved in [Rose, 1970] that this process always produces a chordal graph (see Definition 1.4.11). Finally by adding all edges during the process, we obtain the induced graph (see Figure 1.5h.) We notice three important things from this example: First, the induced graph depends on the elimination order (e.g. if we decided to eliminate C before A, we would have created the edge $A - D$ instead of the edge $B - C$), and for that matter, choosing an elimination order becomes equivalent to choosing a triangulation for the moral graph. Second, for the inference process, we would store in memory all the generated factors, but there are factors that are included in others (e.g. the scope of the factor DE is included in the scope of BDE), it would be convenient in terms of memory and computation time to consider only factors that are not included in other factors i.e. consider only maximal cliques of the induced graph.



(a) A BN. (b) Moral graph. (c) $\sum_A \to ABC$ (d) $\sum_C \to BCD$

(e) $\sum_B \to BDE$ (f) $\sum_D \to DE$ (g) $\sum_E \to EF$ (h) Induced graph.

Figure 1.5: Creating an induced graph from a BN following a VE order.

Definition 1.4.10 (Moral graph) *Being $\mathcal{G} = (\mathbf{X}, \mathcal{E})$ a directed graph, the undirected graph $\mathcal{M}(\mathcal{G}) = (\mathbf{X}, \mathcal{E}')$ is said to be the moral graph of $\mathcal{G}$ iff an edge*

$(X - Y) \in \mathcal{E}'$ exists whenever there exists either $(X \to Y)$ or $(Y \to X)$ or a v-structure $(X \to Z \leftarrow Y)$ in $\mathcal{E}$. It can be also defined simply as a graph with no immoralities (see Definition 1.3.11).

Definition 1.4.11 (Chordal graph) *Being $\mathcal{H} = (\mathbf{X}, \mathcal{E})$ an undirected graph, it is said to be a chordal graph if all its cycles $(X_1, \ldots, X_n), n > 3$ have a chord, i.e. there exists an edge $(X_i - X_j) \in \mathcal{E}$ such that $1 < |i - j| < n - 1$. The process of inserting edges in a graph so that it becomes a chordal graph is called triangulation.*

In Example 1.4.2 we saw the importance of generating the small factors during a VE process. Since minimizing the size of the largest factor is a NP-hard problem [Arnborg et al., 1987], we need at least an algorithm for trying to find approximately the smallest possible factors. The simplest way to achieve this is to choose the variable to eliminate when we are building the induced graph as follows: the variable to eliminate is chosen so that it minimizes a given score function, in a greedy way (see Algorithm 1.2). In [Kjærulff, 1990] there is a discussion of the most common scores used in order to minimize: Min-neighbors (the number of Neighbors), Min-weight (the product of the domain sizes of the variables in the scope of the factors), and Min-fill (number of fill-ins [7] induced by the Elimination).

Algorithm 1.2: Ordering variable elimination by Greedy Search.

Input: $\mathcal{H} = (\mathbf{X}, \mathcal{E})$: an undirected graph, s: a score function.
Output: t: a sequence to perform VE.

```
t ← empty sequence of nodes;
while X is not empty do
    X' ← the set of nodes in X minimizing s;
    X ← a random node in X';
    add X to Sequence t;
    foreach (Y, Z) ∈ Ngb_H(X) × Ngb_H(X), s.t. Y ≠ Z do
        insert edge (Y − Z) into E.
    Remove all edges (X − X') ∈ E whenever X' ∈ X;
    Remove X from X;
return t;
```

In Example 1.4.5 we see how the factors of a variable elimination process are generated for computing $P(F)$. Let's imagine that after that we would like to compute another probability (e.g $P(E, D)$); we could reinitialize the whole process, nonetheless there are some computations that we could re-use; moreover,

[7] The number of fill-ins is the number of edges added to the induced graph whenever a given variable is eliminated.

there are some computations that are independent and can therefore be performed in parallel. For those two reasons, we define graphical structures in the form of clique trees (see Definition 1.4.12), which help us to introduce message-passing algorithms, which perform marginalizations over node's factors and send them as "messages" to other nodes in order to multiply them. This is discussed in details later in this section.

Definition 1.4.12 (Clique graph) *Being $\mathcal{H}$ an undirected graph, $\mathcal{U} = \{\mathcal{C}, \mathcal{E}\}$ is a clique graph of $\mathcal{H}$ if it is an undirected graph whose nodes are cliques $\mathbf{C}_i$ of $\mathcal{H}$, every node of $\mathcal{H}$ is contained in at least one clique of $\mathcal{C}$. Each edge within $\mathcal{U}$ is associated to a separator (see Definition 1.4.13). Whenever that undirected graph is a tree, we call it clique tree.*

Definition 1.4.13 (Separator) *Being $\mathcal{U} = (\mathcal{C}, \mathcal{E})$ a Clique graph, we define $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$ to be a separator of $\mathcal{U}$ iif $(\mathbf{C}_i - \mathbf{C}_j) \in \mathcal{E}$. We consider $\mathbf{S}_{ij}$ and $\mathbf{S}_{ji}$ as two distinct entities even tough they have the same elements.*

We define elimination trees (see Definition 1.4.14), whose nodes correspond to factors cliques generated during the VE process, and whose edges are set s.t. (1) an Elimination tree is always a tree, (2) it verifies the running intersection property (see Definition 1.4.15), which guarantees the consistency between the VE algorithm and the aforementioned message-passing algorithms. In addition, it has the following property: $\forall(\mathbf{C}_i - \mathbf{C}_j) \in \mathcal{E} : \mathbf{C}_j \subset \mathbf{C}_i \iff |Scope(\mathbf{C}_j)| = |Scope(\mathbf{C}_i)| - 1$. This simplifies the task of finding redundant factors [Gonzales, 2008].

Elimination trees give us an easy way to connect cliques, but they are not meant to be used directly to perform "message-passing" algorithms, since they have redundant factors (their cliques are not necessarily maximal cliques of a induced graph). Since elimination trees respect the running intersection property (which ensures that all cliques containing a given variable X are always in a connected sub-graph). Then we use them to build junction trees (JTs, see Definition 1.4.16), which have no redundant cliques [8]; this is done by applying Algorithm 1.3. Once having a JT, we have to assign every factor of the inference problem (CPTs and evidence) to exactly one clique; this is done by assigning each factor to any clique being a superset of that factor (see Definition 1.4.17). The JT created from a given BN is not unique, it depends of the VE order (or equivalently

[8] In the literature, it is often proposed to build JTs as the result of applying a maximum spanning tree algorithm over a clique graph composed of induced graph's maximal cliques, weighting each edge as $W_{(\mathbf{C}_i - \mathbf{C}_j)} = |\mathbf{C}_i \cap \mathbf{C}_j|$. Finding the maximal cliques is often feasible, but we must recall it is a NP-hard problem [Karp, 1972].

to the triangulation of the induced graph), and it happens that there are JTs that are more convenient that others. The measure of how convenient a JT is is its width, i.e. the size of the largest factor (the smallest, the better); we call the width of the best possible JT of a BN the treewidth of a BN (or a graph in general, see Definition 1.4.18); finding it is equivalent to finding the optimal VE order, which is NP-hard. It is proved in [Arnborg et al., 1987] that the inference complexity is exponential in the treewidth of a BN.

Definition 1.4.14 (Elimination Tree) *Being $(X_1, \ldots, X_n)$ a VE order and $(\mathbf{C}_1, \ldots, \mathbf{C}_n)$ the factors (induced graph's cliques) generated during a VE process respecting that order, an elimination tree is defined as the clique tree $\mathcal{T} = \{\mathcal{C}, \mathcal{E}\}$ where:*

- $\mathcal{C} = \{\mathbf{C}_i : i \in \{1, \ldots, n\}\}$
- $\mathcal{E} = \{(\mathbf{C}_i - \mathbf{C}_j) : 1 \leq i < n, j = \min\{k \neq i : X_k \in \mathbf{C}_i\}\}$

Definition 1.4.15 (Running Intersection Property) *Being $\mathcal{H} = (\mathcal{C}, \mathcal{E})$ a clique graph, we say that $\mathcal{H}$ respects the running intersection property iff for any variable X contained in two cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ in $\mathcal{H}$, X belongs also in every clique of any path $(\mathbf{C}_i, \ldots, \mathbf{C}_j)$ in $\mathcal{H}$, and there exists at least one path between $\mathbf{C}_i$ and $\mathbf{C}_j$.*

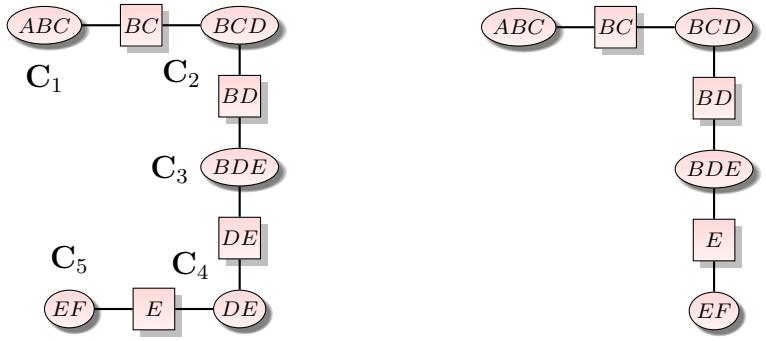
Definition 1.4.16 (Junction Tree) *A Junction Tree (JT) is a clique tree verifying the running intersection property, having no clique being a subset of another clique in it.*

Algorithm 1.3: Creating a junction tree from an elimination tree.

Input: $\mathcal{T} = (\mathcal{C}, \mathcal{E})$: an elimination tree.
Output: A junction tree.

```
1 mark all edges of E as false;
2 for i = (n, ..., 1) do
3     foreach C_j st. j > i, and C_j ⊂ C_i, and (C_j − C_i) ∈ E, and
        (C_j − C_i) is marked as false, do
4         foreach C_k ∈ Ngb_T(C_i) st. C_k ≠ C_j do
5             insert edge E = (C_k − C_j) into E;
6             mark E as true if k > i. Otherwise, mark it as false;
7         remove C_i from C, and all its adjacent edges in E;
8 return t;
```

Definition 1.4.17 (Valid Clique Assigment) *Given a JT $\mathcal{T} = (\mathcal{C}, \mathcal{E})$, we say a factor ϕ is a valid assignment for a clique $\mathbf{C} \in \mathcal{C}$ iff $Scope(\phi) \subseteq \mathbf{C}$.*



(a) An elimination tree (b) A junction tree

Figure 1.6: How to build a JT from a elimination tree.

Definition 1.4.18 (Width and Tree Width of a JT) *We define the width of a JT as size of its largest clique minus 1, whereas we define the tree width of a graph $\mathcal{K}$ as the minimum width of all its possible JTs.*

Example 1.4.6 *We take the BN from Example 1.4.5, whose inducted graph was created. From that, we construct a Elimination Tree using directly Definition 1.4.14, which is shown in Figure 1.6a. Then, from that elimination tree, we construct a JT applying the Algorithm 1.3, which is shown in Figure 1.6b.*

Once we have constructed a JT, we can describe the Shafer-Shenoy algorithm [Shenoy and Shafer, 1990], which has three steps: (a) Valid assignment of all cliques, (b) Collect step, and (c) diffuse step. We give the algorithm as input a JT, a set of factors (CPTs and evidences in the BN case). At the end of the algorithm, for each clique $\mathbf{C}_i$, the product of the factors stored in $\mathbf{C}_i$ by the messages sent toward $\mathbf{C}_i$ is the joint posterior distribution of the variables of $\mathbf{C}_i$. This algorithm does not have a particular target, so if we need the marginal distribution of any variable it is needed to additionally to marginalize the joint posterior of any

clique that contains it. This process is described in detail in Algorithm 1.4.

Algorithm 1.4: Shafer-Shenoy Algorithm.

Input: $\mathcal{T}$: a JT, Φ: a set of factors, $\mathbf{e}$: a set of evidence

foreach *Clique* $\mathbf{C}$ *in* $\mathcal{T}$ **do**
  $factor\,[\mathbf{C}] \leftarrow \mathbb{1}$;
foreach *Separator* $\mathbf{S}_{ij} \in \mathcal{T}$ **do**
  $factor\,[\mathbf{S}_{ij}] \leftarrow \mathbb{1}$;
  $factor\,[\mathbf{S}_{ji}] \leftarrow \mathbb{1}$;
foreach *Factor* $\phi \in \Phi \cup \mathbf{e}$ **do**
  `// Valid clique assignment`
  $\mathbf{C} \leftarrow$ a clique in $\mathcal{T}$ s.t. $Scope(\phi) \subseteq \mathbf{C}$;
  $factor\,[\mathbf{C}] \leftarrow factor\,[\mathbf{C}] \times \phi$;
$\mathbf{C}_r \leftarrow$ a clique in $\mathcal{T}$; `// root clique`
foreach $\mathbf{C}_i \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r)$ **do**
  $Collect(\mathbf{C}_i, \mathbf{C}_r)$;
foreach $\mathbf{C}_i \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r)$ **do**
  $\phi \leftarrow factor\,[\mathbf{C}_r]$;
  foreach $\mathbf{C}_j \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r) \setminus \{\mathbf{C}_i\}$ **do**
    $\phi \leftarrow \phi \times factor\,[\mathbf{S}_{jr}]$;
  $factor[\mathbf{S}_{ri}] \leftarrow \sum_{\mathbf{C}_r \setminus \mathbf{S}_{ri}} \phi$;
  $Diffuse(\mathbf{C}_i, \mathbf{C}_r)$;

Subroutine 1.5: Shafer-Shenoy collect step.

Collect $(\mathbf{C}_r, \mathbf{C}_s)$
  $\phi \leftarrow factor\,[\mathbf{C}_r]$;
  if $\mathbf{C}_r$ *is not a leaf in* $\mathcal{T}$ **then**
    foreach $\mathbf{C}_i \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r) \setminus \{\mathbf{C}_s\}$ **do**
      $Collect(\mathbf{C}_i, \mathbf{C}_r)$;
      $\phi \leftarrow \phi \times factor\,[\mathbf{S}_{ir}]$;
  $\mathbf{X} \leftarrow Scope(\phi) \setminus \mathbf{S}_{rs}$;
  $factor\,[\mathbf{S}_{rs}] \leftarrow \sum_{X \in \mathbf{X}} \phi$;

Example 1.4.7 (Shafer-Shenoy) *In this example we use the Shafer-Shenoy algorithm for the JT from Figure 1.6b, which is made from the BN from Figure 1.5a. In this context, we denote ϕ_i to be the factor associated with the clique $\mathbf{C}_i$, such that $Scope(\phi_i) = \mathbf{C}_i$. We also denote ϕ_{ij} to be associated to the separator $\mathbf{S}_{ij}$,*

Subroutine 1.6: Shafer-Shenoy diffuse step.

Diffuse $(\mathbf{C}_r, \mathbf{C}_s)$
  foreach $\mathbf{C}_i \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r) \setminus \{\mathbf{C}_s\}$ **do**
    $\phi \leftarrow factor\,[\mathbf{C}_r]$;
    foreach $\mathbf{C}_j \in \mathbf{Ngb}_{\mathcal{T}}(\mathbf{C}_r) \setminus \{\mathbf{C}_i\}$ **do**
      $\phi \leftarrow \phi \times factor\,[\mathbf{S}_{jr}]$;
    $\mathbf{X} \leftarrow Scope(\phi) \setminus \mathbf{S}_{ri}$;
    $factor\,[\mathbf{S}_{ri}] \leftarrow \sum_{X \in \mathbf{X}} \phi$;
  $Diffuse(\mathbf{C}_i, \mathbf{C}_r)$;

such that $Scope(\phi_{ij}) = \mathbf{S}_{ij}$. *It is worth noticing that we distinguish* ϕ_{ij} *and* ϕ_{ji}. *First, we make a valid clique assignment for all nodes (see Figure 1.7):*

$$\phi_1 = P(A)P(B|A)P(C|A) = P(A)P(B,C|A) = P(A,B,C)$$
$$\phi_2 = P(D|C)$$
$$\phi_3 = P(E|B,D)$$
$$\phi_4 = P(F|E)$$

In the valid assignment of ϕ_1, *the result is obtained by using the conditional independence* $P(B|A)P(C|A) = P(B,C|A)$, *which is encoded in the original BN. There are two simple rules for the message-passing: (1) before sending a message to a neighbor, a clique must wait until receiving messages from all its other neighbors; and (2) the message from a clique* $\mathbf{C}_i$ *to a Clique* $\mathbf{C}_j$ *is the product of the potential associated to the clique* $\mathbf{C}_i$ *and all the messages it received from other cliques, marginalized over the variables of* $\mathbf{C}_i \setminus \mathbf{C}_j$, *i.e.* $\phi_{ij} = \sum_{\mathbf{C}_i \setminus \mathbf{C}_j} \phi_i \times \prod_{k \neq j} \phi_{ki}$. *We choose the clique* EF *as root clique, and we proceed with the collection step for the message-passing:*

$$\phi_{12} = \sum_A \phi_1 = \sum_A P(A,B,C) \qquad = P(B,C)$$
$$\phi_{23} = \sum_C \phi_2 \times \phi_{12} = \sum_C P(D,C)P(B|C,D) = P(B,D)$$
$$\phi_{34} = \sum_{B,D} \phi_3 \times \phi_{23} = \sum_{B,D} P(E|B,D)P(B,D) = P(E)$$

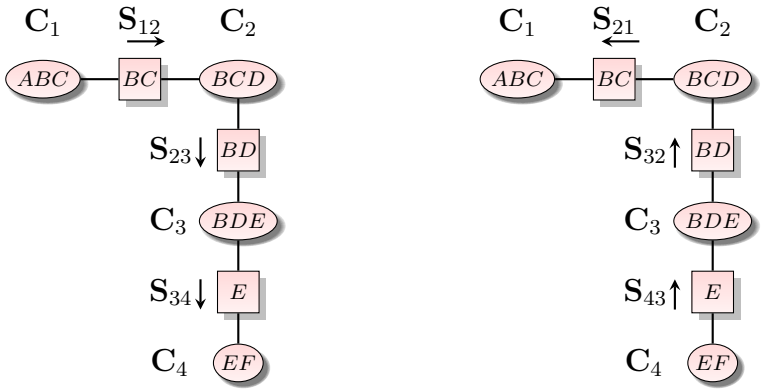
For finding the result of ϕ_{23} *result we observe that* $P(D|C)P(B,C) = P(D,C)P(B|C)$, *and that* B *and* C *are independent in the original BN,* $P(B|C) = P(B|C,D)$. *We proceed with the diffuse step:*

$$\phi_{43} = \sum_F \phi_4 = \sum_F P(F|E) \qquad = \mathbb{1}_E$$

$$\phi_{32} = \sum_E \phi_3 \times \phi_{43} = \sum_E P(E|B,D) = \mathbb{1}_{BD}$$
$$\phi_{21} = \sum_D \phi_2 \times \phi_{32} = \sum_D P(D|C) \quad = \mathbb{1}_{BC}$$

Finally we verify that the posterior distribution of all cliques' variables is the product of the potential of those cliques and all the messages they received:

$$\begin{aligned}
P(A,B,C) &= \phi_1 \times \phi_{21} &&= P(A,B,C) \times \mathbb{1}_{BC} \\
P(B,C,D) &= \phi_2 \times \phi_{12} \times \phi_{32} &&= P(D|C) \times P(B,C) \times \mathbb{1}_{BD} \\
P(B,D,E) &= \phi_3 \times \phi_{23} \times \phi_{43} &&= P(E|B,D) \times P(B,D) \times \mathbb{1}_{E} \\
P(E,F) &= \phi_4 \times \phi_{34} &&= P(F|E) \times P(E)
\end{aligned}$$



(a) Collect step. (b) Diffuse step

Figure 1.7: Message-passing for the Shafer-Shenoy algorithm.

1.5 Learning

BNs can be constructed "by hand" by means of experts' knowledge; nonetheless, this approach has several caveats, such as its limitation to problems where the number of random variables is small, and the fact that it requires a large amount of time from an expert, which sometimes is expensive. In addition, there exist systems that are not fully understood by experts; moreover, even if an expert has a good understanding of a system, he would inevitably encode his own personal biases into a BN. On the other hand, in many domains we have lots of information in the form of data records, often representing information that is not part of any domain's expert knowledge. Consequently, it is in our best interest to construct

BNs from existing databases rather than from experts' knowledge; this approach is called learning.

Learning BNs consists in estimating their structure (DAGs) and their parameters (CPTs). As we will see, estimating the CPTs is in general an easy task whenever a BN structure is given: it is computed directly from a close form formula (under certain assumptions that will be discussed later). On the other hand, finding the BN structure from data is a NP-hard task [Chickering, 1996]. Many approximate algorithms exist. They can be roughly divided into three broad categories: Constraint-based approaches, Score-based approaches and hybrid approaches.

1.5.1 Constraint-based approaches

Constraint-based approaches exploit the meaning of the edges in a BN structure (or more precisely the lack of edges) in the BN: the joint probability of the random variables being equal to the product of the conditional probabilities of each random variable given its parents in the graphical structure, the lack of an edge between two nodes represents a conditional independence between the corresponding random variables. Therefore, using statistical conditional independence tests (typically χ^2 or G^2), it is possible to determine those edges that shall not belong to the BN. The orientation of those edges is assigned in a way such that it is consitent with the set of independences found out from the independence tests, by means of a set of deterministic rules [Meek, 1995a]. Graphs obtained by the constraint-based approaches rely heavily on the test significance parameter α: the smaller the significance, the fewer the edges in the BN structure learnt.

The two most important algorithms in this approach are the IC algorithm [Pearl and Verma, 1991, Pearl, 2000] and the PC algorithm [Spirtes et al., 2001]. The IC algorithm uses the independence tests to add edges into a graph which is initially empty, which actually models a minimal D-map (see Definition 1.3.8); whereas the PC algorithm uses those tests to remove edges from an initially complete graph, which tries to model a minimal I-map. Under ideal faithfulness conditions (see Definition 1.3.9), both D-maps and I-maps are equivalent. In practice we prefer I-maps, which can lead us to more compact representations, so the PC algorithm and its variants [Abellán et al., 2006] are the most widely used. The PC algorithm has a caveat: it is very sensitive to the order in which conditional independence tests are performed [Fast et al., 2008], this problem is solved in [Colombo and Maathuis, 2014] which proposes the PC-stable algorithm, which in addition allows better parallelization of independence tests.

Conditional independence tests of $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$ are known to be inaccurate whenever the size of the dataset is not big enough, and this is especially true when the conditioning set $\mathbf{Z}$ has many elements [Tsamardinos et al., 2003, Spirtes et al., 2001, Dash and Druzdzel, 2012], which is particularly inconvenient since wrongly

removed edges can cause compensatory errors in later iterations. In order to alleviate this problem, there are many algorithms that perform the tests with smaller conditional sets, just like [Steck and Tresp, 1999] which restricts the variables of $\mathbf{Z}$ to only those variables having paths linking X and Y in the current iteration's graph $\mathcal{G}$. In [Abellán et al., 2006] it is proposed that if the size of $\mathbf{Z}$ is bigger than the minimum cutset of X and Y [9] we should use that minimum cutset as conditioning set instead of $\mathbf{Z}$. Also, there are the MMPC [Tsamardinos et al., 2003] and the fast-IAMB [Yaramakala and Margaritis, 2005] algorithms, which seek the set of parents-and-children and the Markov blanket (respectively) of each variable in a depth-first manner.

In [Cheng et al., 2002], there is a three phase algorithm for learning BNs. Its three phases are drafting, thickening and thinning. The drafting phase quickly obtains an initial graph representation as a tree by means of the algorithm proposed in [Chow and Liu, 1968]. The thickening phase adds edges to the graph obtained in the preceding phase whenever their two extremal nodes cannot be declared as independent using conditional independence tests. The thinning phase does the opposite of the preceding phase, it removes any edge whenever its nodes are determined to be independent by independence tests. Although this method has been criticized for its monotone DAG assumption [Chickering and Meek, 2006], it is widely used in practice.

1.5.2 Search-based approaches

Search-based approaches [Cooper and Herskovits, 1992a, Heckerman, 1995] consider that the best structure for the BN is the one that has the highest likelihood given the data (that we shall denote by $\mathcal{D}$), i.e., they look for $\mathcal{G}^* = \text{Argmax}_{\mathcal{G}}\, P(\mathcal{G} \mid \mathcal{D})$. Since there is a super-exponential number of DAGs $\mathcal{G}$ with n nodes, a neighborhood search method is usually defined. Any neighborhood method has to take four things into account: the search space, the neighborhood operators used to explore it, the search strategy and a score to compare the likelihood of each candidate solution. The neighborhood search methods typically start from a given graph $\mathcal{G}^0$ (often an empty graph) and try to find in its neighborhood another graph $\mathcal{G}$ with a higher likelihood. They iterate the process, starting from the new graph until a local maximum is reached.

The natural and most used search space is that of DAGs $\mathcal{G}$, whose neighborhood is often defined as the set of graphs resulting from one of the following three neighbor operators: the addition of a new arc to $\mathcal{G}$, the removal of an arc from $\mathcal{G}$ or

[9] The minimum cutset of X and Y in $\mathcal{G}$, is the minimal set of nodes whose removal in $\mathcal{G}$ would cause that there exists no path between X and Y. They can be computed with the algorithm proposed in [Acid and De Campos, 1996].

the reversal of an arc in $\mathcal{G}$. However, there are other DAGs neighborhood operators that have been successfully used to avoid certain local maxima when searching the DAG space, like [Moore and Wong, 2003] which defines an additional operations which reorientates all adjacent edges of a given node in a optimal manner. There is also [De Campos et al., 2002] which redefines the reversal operator, removing all common parents of its two concerned nodes and adding as new parents a subset of the removed ones after reverting the edge. [Vandel et al., 2012] introduces a swapping operator which, in a single operation, removes an edge connecting a couple of nodes and adds an edge into a different couple. Since different DAGs can encode the same independences, there are also methods that define the Markov Equivalence Classes (in the form of CPDAGs) as search spaces, which can be considerably smaller than DAGs search spaces, in fact [Gillispie and Perlman, 2001] shows empirically that a CPDAG space is asymptotically 3.75 times smaller than a DAG space for the same number of nodes; the most notable example of this is in [Chickering, 1995, Chickering, 2002a] where he defines six neighborhood operations to explore the CPDAG space: insert/remove of directed/undirected edges, revert directed edges, and add V-structure. There is another common less obvious search space, which is the topological order (see Definition 1.2.12). This idea was developed in [Teyssier and Koller, 2012] where the fact that learning a BN knowing its topological order has the polynomial complexity $\mathcal{O}(n^k)$[10] (instead of being NP-Hard) is exploited to swap random variables positions in the topological order to execute many learning procedures.

The most common search strategy in the DAG space is the greedy search, which is often improved by using tabu-lists [Glover, 1990], which after any neighbor operation for a couple of nodes, forbids any other operation related to those nodes during a given number of iterations. The greedy search is also often complemented with random restarts and noise introduction in order to escape local maxima [Elidan et al., 2002]. During the iterations of those algorithms, after applying any neighbor operator, we must verify that the resulting graph $\mathcal{G}$ is still a DAG (if not it is not a valid operator), this operation can be expensive, that is why in [Cooper and Herskovits, 1992b] there is a particular case of greedy search: the K2 algorithm, which uses a fixed topological order, having the advantages of not requiring any reverting-edge operator nor any DAGs verification during its iterations, and thus having a much faster execution than other mentioned greedy search algorithms. There is also a greedy search algorithm called Greedy Equivalent Search [Chickering, 2002a, Chickering, 2002b] which explores the CPDAG space with its six neighborhood operators mentioned in the preceding paragraph, and an alternative approach that explores the CPDAG space using only addition and removal operators.

[10] being k the maximum number of parents in the learned $\mathcal{G}$ for each of its n nodes

The likelihoods of each DAG $\mathcal{G}$ through the search are computed through different scores. There are two main types of score: Bayesian score functions and information-theoretic functions. The Bayesian score functions evaluate the posterior probability of data for a given candidate model, using a defined prior probability distribution: the main examples of this type of score are the BD [Heckerman et al., 1995], BDe [Heckerman et al., 1995], BDeu [Buntine, 1991]; K2 [Cooper and Herskovits, 1992a], *etc*. Most of them differ essentially from the *a priori* hypotheses they assume. In the other hand, information-theoretic scores evaluate models according to entropy-based criteria measuring how well a current model can be used to encode compactly its learning dataset. The most used information-theoretic scores are the log-likelihood (which is by itself a direct measure of the number of bits needed to represent the learning data), AIC [Akaike, 1970], BIC/MDL [Schwarz, 1978, Lam and Bacchus, 1994], MIT [Campos, 2006], NML [Silander et al., 2008, Silander et al., 2018]. In the present work we will focus on search-based approaches so that whenever we talk about learning BNs, unless otherwise specified, we will refer to this approach.

1.5.3 Hybrid approaches

As mentioned before, constraint-based approaches can be inaccurate because of compensatory errors due to independence tests, while search-based approaches tend to find suboptimal solutions due to the huge search spaces they explore (typically DAGs). We call hybrid approaches the ones that combine the best part of both previously mentioned approaches, usually exploiting constraint-based approaches to quickly determine an initial graph to be improved by a search-based approach or to reduce considerably the search space.

The first important hybrid method to learn BNs was proposed in [Friedman et al., 1999], which has two phases: Restrict and Maximize. In the restrict phase, for every couple of variables (X, Y) they measure the Kullback-Leibler divergence (DKL) between $P_{\mathcal{G}}(X, Y)$ and $P_{\mathcal{G}}(X)P_{\mathcal{G}}(Y)$ w.r.t. a current candidate graph $\mathcal{G}$. For each variable, the other k variables corresponding to the biggest DKLs are marked as potential parents of X [11]. In the maximize phase, a Hill Climbing search algorithm is executed to improve the current candidate graph $\mathcal{G}$, allowing only arcs respecting the potential parents lists created in the restrict phase, which gives as a result sparse graphs. These two phases are repeated iteratively until convergence. To this day, this algorithm is one of the very few that can be used to learn BNs with thousands of random variables.

There are many other state-of-the-art hybrid approaches, like [Tsamardinos

[11] The DKL between $P(X, Y)$ and $P(X)P(Y)$ measures how divergent those two distributions are, and in this particular case it is a measure of the independence between X and Y.

et al., 2006] which proposes the Min-Max Hill Climbing algorithm, which precomputes a parents candidate set for each random variable via independence tests (using the MMPC algorithm [Tsamardinos et al., 2003]), in order to perform a greedy hill climbing search with tabu-lists, using the BDeu score. Another popular hybrid approach is proposed in [van Dijk et al., 2003], which starts performing the PC algorithm (limited to independence tests of zero and first order [12]), in order to find a good initial DAG, to be improved through a hill climbing search, forbidding arcs whenever independences are found, thus reducing the search space.

In [Schulte et al., 2009], they propose a search algorithm that reduces the DAG space by testing the conditional independences of the form $(X \perp\!\!\!\perp Y \mid \mathbf{MB}_{\mathcal{G}}(X))$, only allowing to add edges whenever their involved variables do not verify that independence assertion. In every iteration, a neighbor operation must be performed in order to maximize the global score, continuing even if the best neighbor operation decreases it, and not stopping its iterations until the independence tests do not propose any new potential edges to add. Later, [Gámez et al., 2011] proposed the Constraint Hill Climbing algorithm, which performs computations of DAG space reduction and hill climbing search iterations at the same time. This is done by adding any variable Y to a forbidden-parents list for every variable X whenever adding the edge $(Y \rightarrow X)$ decreases the search score, or whenever removing it increases the score, and taking arc reversals as a removal followed by an addition. Once the search algorithm finds a locally-optimal DAG, the forbidden-parents lists are emptied and the Constrained Hill Climbing process is restarted, using the obtained graph as the initial one [13], in order to avoid sub-optimal solutions.

More recently, [Liu et al., 2017] made a great theoretical contribution with the following theoretical result: being a random variable set $\mathcal{X}$ the union of three disjoint sets $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$, letting $\mathcal{G} = (\mathcal{X}, \mathbf{E})$, $\mathcal{G}_1 = (\mathbf{X} \cup \mathbf{Z}, \mathbf{E}_1)$ and $\mathcal{G}_2 = (\mathbf{Y} \cup \mathbf{Z}, \mathbf{E}_2)$ be DAGs faithful to the distributions $P(\mathcal{X})$, $P(\mathbf{X}, \mathbf{Z})$ and $P(\mathbf{Y}, \mathbf{Z})$. If the d-separation $(\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \mathbf{Z})$ holds in $\mathcal{G}$, then graph $\mathcal{G}' = (\mathcal{X}, \mathbf{E}')$ is proved to be equivalent to $\mathcal{G}$ if $\mathbf{E}' = (\mathbf{E}_1 \cup \mathbf{E}_2) \setminus \{(X_i, X_j) \mid X_i, X_j \in \mathbf{Z} \wedge (X_i, X_j) \notin \mathbf{E}_1 \cap \mathbf{E}_2\}$ [14]. This result is used to propose a hybrid approach in two phases: separation and reunion. The separation phase uses conditional independence tests to find successive partitions of a random variable sets, noting them in a binary tree. Then any learning algorithm can be used to quickly learn many small DAGs. In the reunion phase, the learned DAGs are merged using the theorem discussed in the beginning of this paragraph.

[12] *e.g.*, testing $(X \perp\!\!\!\perp Y \mid \mathbf{Z})$, only for $\mathbf{Z}$ s.t. $|\mathbf{Z}| \in \{0, 1\}$

[13] They actually propose the possibility to restart the process many times, but already show good experimental results with only one single restart.

[14] If $\mathbf{E}'$ presents v-structures not present in $\mathcal{G}_1$ or $\mathcal{G}_2$, edge reversals must be performed s.t. $\mathcal{G}'$ contains exactly the same v-structures as $\mathcal{G}_1$ and $\mathcal{G}_2$, which is proved to be always attainable.

1.5.4 Exact approaches

Exact approaches seek to find optimal BN structures: the ones globally maximizing a score function (and not only locally as in search-based approximations). Learning BN structures is a NP-hard problem [Chickering, 1996], but not in the particular case when we limit number of parents for each node to one. The solution for this particular case is proposed in [Chow and Liu, 1968], by computing a maximum-spanning tree over a complete graph, having as edge weights the mutual information between its adjacent nodes (computed from a learning dataset). The obtained undirected maximum-spanning tree is transformed into a DAG by choosing a root node, and setting all edges' directions outward from it. This process is guaranteed obtain the structure that maximizes the likelihood w.r.t. the learning data, while having polynomial complexity: $\mathcal{O}(n^2 N)$, for n variables and N records in the learning dataset.

For general cases, when nodes are allowed to have two or more parents, exact learning BN structures is proved to be NP-hard [Dasgupta, 1999]. There are two approaches that can obtain solutions to this problem in exponential time (which is not as bad as it could be, since DAG searching spaces have super-exponential sizes). Those two approaches are dynamic programming and branch-and-bound.

The use of dynamic programming to find the optimal BN structure was (to the best of our knowledge) first proposed by [Ott et al., 2003] and [Koivisto and Sood, 2004]: both proposed independently similar procedures where the optimal solution can be found with $\mathcal{O}(n2^n)$ dynamic programming iterations. In [Singh and Moore, 2005] and [Silander and Myllymaki, 2012] improvements to those algorithms are proposed, so that their complexity is reduced to $\mathcal{O}(n2^{n-1})$ and $\mathcal{O}(n^2 2^{n-2})$, respectively. The main caveat of dynamic programming approaches is the amount of memory required to execute them (and not their execution times), for instance, the algorithm proposed in [Silander and Myllymaki, 2012] requires 2^{n+2} bytes of RAM and $12n2^{n-1}$ bytes of hard disk space, which can be prohibitive (*e.g.*, whenever $n = 40$, it would require 4.39 and 264 terabytes, respectively).

Branch-and-bound (B&B) approaches seek to find exact solutions while trying to explore as little as possible the search space, by discarding important parts of it thanks to easy-to-compute upper bounds they might have. While B&B was already used for learning BN structures in early works [Dasgupta, 1999], the first method to successfully use it to find globally optimal solutions is proposed in [De Campos et al., 2009, Campos and Ji, 2011], when they analyze mathematical properties of scores AIC, BIC and BD, in order to provide upper bounds to discard parents sets without directly inspecting them; and they also propose a search framework that supports the addition of structural constrains in the form of forced edges, maximum number of parents, and *or/not* logical op-

erations with the two preceding types of constraint. In [Jaakkola et al., 2010], they represent the valid acyclic structures space through a polytope, so the BN structure can be found using integer programming methods with linear programming relaxations, iteratively tightening bounds by searching new constraints. In a similar manner, [Cussens, 2012, Cussens et al., 2017] propose another integer-programming-based method, which finds upper bounds in the search space by using linear programming relaxation and by not imposing acyclicity constraints. The effectiveness of this method relies heavily in finding good cutting planes, for which the authors provide some criteria.

1.5.5 Problem statement

In this part we introduce the notations required to address the problem of BN learning, the assumptions we make to approach it and its formal statement. We limit ourselves to learn discrete variables with a (small) finite domain. In a nutshell, the problem of learning consists in finding the Bayesian network $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ that best fits a dataset $\mathcal{D}$. $\mathcal{B}$ is defined over a BN structure $\mathcal{G}$ (a DAG) and a set of parameters $\mathbf{\Theta}$, with $\mathcal{G} = (\mathbf{X}, \mathcal{E})$. We define n as the number of random variables in $\mathcal{G}$, *i.e.* $n = |\mathbf{X}|$, so that $\mathbf{X} = \{X_i\}_{i=1}^{n}$. The parameters $\mathbf{\Theta} = \{\theta_i\}_{i=1}^{n}$ are defined s.t. θ_i corresponds to the CPT associated with $P_{\mathcal{B}}(X_i|\mathbf{Pa}(X_i))$. For each X_i, we name r_i and q_i to the domain sizes of itself and its parents, respectively, *i.e.* $r_i = |\Omega_{X_i}|$ and $q_i = |\Omega_{\mathbf{Pa}(X_i)}|$ [15]. We refer to the domain values $\Omega_{X_i} = \{x_{ik}\}_{k=1}^{r_i}$ and $\Omega_{\mathbf{Pa}(X_i)} = \{\mathbf{pa}(X_i)^{(j)}\}_{j=1}^{q_i}$. Then, the CPT for the node X_i is represented as $\theta_i = \{\theta_{ij}\}_{j=1}^{q_i}$, where $\theta_{ij} = \{\theta_{ijk}\}_{k=1}^{r_i}$ represents the probability distribution $P_{\mathcal{B}}(X_i|\mathbf{Pa}(X_i) = \mathbf{pa}(X_i)^{(j)})$, and θ_{ijk} represents the probability value $P_{\mathcal{B}}(X_i = x_{ik}|\mathbf{Pa}(X_i) = \mathbf{pa}(X_i)^{(j)})$. Letting N be the number of records of the dataset, then $\mathcal{D} = \{\mathbf{x}^{(m)}\}_{m=1}^{N}$, where $\mathbf{x}^{(m)}$ is an instantiation of $\mathbf{X}$, which for the purposes of this chapter we will assume to be complete (with no missing values).

We make the following assumptions when learning a BN $\mathcal{B} = (\mathcal{G}, \Theta)$ from a dataset $\mathcal{D}$:

1. **Faithfulness**. The Underlying distribution P that generates $\mathcal{D}$ can be represented as a P-map of a graph $\mathcal{G}$ (see Definition 1.3.9).

2. **Completeness** (or causal sufficiency). For any couple of random variables in $\mathcal{G}$, there is no hidden (latent, or unobserved) variable being a common parent of those in $\mathcal{G}$.

3. **Independent and identically distributed records.** (iid.) All records are generated by the same distribution, and all record's values are influenced

[15] By abuse of notation, if $\mathbf{Pa}(X_i) = \emptyset$, then $q_i = 1$.

only by the generating distribution and not by other records: for any model $\mathcal{B}$, $P(\mathcal{D}|\mathcal{B}) = \prod_{m=1}^{N} P(\mathbf{x}^{(m)}|\mathcal{B})$.

4. **Globally independent parameters.** The values of Θ have no influence one to another: $\pi(\mathbf{\Theta}) = \prod_{i=1}^{n} \pi(\theta_i)$, where $\pi(\cdot)$ represents an *a priori* distribution.

5. **Locally independent parameters.** The parameters for Node X_i are independent for different instantiations of its parents: $\pi(\theta_i) = \prod_{i=1}^{q_i} \pi(\theta_{ij})$.

6. **Parameters modularity.** For two BNs defined over two different graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, if $\mathbf{Pa}_{\mathcal{G}_1}(X_i) = \mathbf{Pa}_{\mathcal{G}_2}(X_i)$, then both BNs shall have the same value for θ_i.

7. **Dirichlet prior** The prior distribution $\pi(\theta_{ij})$ follows a Dirichlet distribution, defined over the hyper-parameters $\mathbf{A} = \{\alpha_k\}_{k=1}^{r_i}$:

$$Dirichlet(\theta_{ij}|\alpha_1, \ldots, \alpha_{r_i}) = \frac{\Gamma(\sum_{k=1}^{r_i} \alpha_k)}{\prod_{k=1}^{r_i} \Gamma(\alpha_k)} \prod_{k=1}^{r_i} \theta_{ijk}^{\alpha_k - 1}$$

being $\Gamma(\cdot)$ the gamma function. This assumption is particularly convenient because, as we will see latter in this chapter, the likelihood function $P(\theta_{ij})$ follows a multinomial distribution, for which the Dirichlet distribution turns out to be the conjugate prior, which simplifies the task of performing a Bayesian parameter estimation.

With those assumptions, we can now define the BN learning problem as finding the BN $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ that maximizes a likelihood measure of $L(\mathcal{B} : \mathcal{D})$. A natural way to define that likelihood is to evaluate the probability of the learning dataset $\mathcal{D}$ wrt. $\mathcal{B}$:

$$\mathcal{B}^* = (\mathcal{G}^*, \mathbf{\Theta}^*) = \underset{\mathcal{B}}{\text{Argmax}}\, L(\mathcal{B} : \mathcal{D}) = \underset{(\mathcal{G}, \mathbf{\Theta})}{\text{Argmax}}\, P(\mathcal{D}|\mathcal{G}, \mathbf{\Theta})$$

The preceding equation suggests that we could search for the optimal BN $\mathcal{B}^*$ in the space of $\mathcal{G}, \mathbf{\Theta}$. Nevertheless, the parameters modularity assumption tells us that for a given $\mathcal{G}$ there is only one possible value of $\mathbf{\Theta}$, so that we can express the above mentioned likelihood as $P(\mathcal{D}|\mathcal{G}, \Theta) = P(\mathcal{D}|\mathcal{G})$. This leads to the conclusion that the BN learning can be done in two separate steps: learning the structure $\mathcal{G}$ and estimating the parameters $\mathbf{\Theta}$ (once for a given structure).

1.5.6 Maximum likelihood estimation of parameters

The most direct way to estimate the parameters Θ for a given DAG $\mathcal{G}$, is to choose them s.t. they maximize the likelihood of $\mathcal{D}$:

$$\mathbf{\Theta}^{MLE} = \underset{\mathbf{\Theta}}{\text{Argmax}}\, P(\mathcal{D}|\mathcal{G}, \mathbf{\Theta})$$

Where MLE stands for maximum likelihood estimation. Thanks to the iid. assumption we can decompose the likelihood as follows:

$$\begin{aligned} P(\mathcal{D}|\mathcal{G}, \mathbf{\Theta}) &= \prod_{m=1}^{N} P(\mathbf{x}^{(m)}|\mathcal{G}, \mathbf{\Theta}) \\ &= \prod_{m=1}^{N}\prod_{i=1}^{n} P(X_i = x_{ik_m}|\mathbf{Pa}(X_i) = \mathbf{pa}(X_i)^{(j_m)}, \mathbf{\Theta}) \\ &= \prod_{m=1}^{N}\prod_{i=1}^{n} \theta_{ij_mk_m} \end{aligned}$$

Where x_{ik_m} and $\mathbf{pa}(X_i)^{(j_m)}$ represent the value taken by X_i and $\mathbf{Pa}(X_i)$ in the m-th record of $\mathcal{D}$ respectively. We can express that product in a more convenient way, letting N_{ijk} be the number of times the parameter θ_{ijk} is multiplied in the precedent equation:

$$P(\mathcal{D}|\mathcal{G}, \mathbf{\Theta}) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i} (\theta_{ijk})^{N_{ijk}} \tag{1.13}$$

In order to maximize Equation 1.13 we present the following theorem:

Theorem 1.5.1 *Being $\mathbf{A} = \{A_i\}_{i=1}^{N}$ a set of real valued variables with positive domain, s.t. $\sum_i A_i = k$, being k a real valued positive constant, and letting $\mathbf{b} = \{b_i\}_{i=1}^{N}$ be a set real valued positive constants, then the product $\prod_{i=1}^{N} A_i^{b_i}$ takes its maximum value whenever $A_i = kb_i / \sum_i b_i$.*

Proof. Expressing any value $A_j \in \mathbf{A}$ as $A_j = k - \sum_{i\neq j} A_i$ and replacing it into the product, so that it becomes a function of $n-1$ variables: $f(\mathbf{A} \setminus \{A_j\}) = (\prod_{\substack{i=1\\ i\neq j}}^{N} A_i^{b_i})(k - \sum_{i\neq j} A_i)^{b_j}$, then all values $A_i \neq A_j$ are obtained by solving the equation $\frac{\partial f}{\partial A_i} = 0$, while A_j can be obtained by difference. Solving those equations, we obtain the presented result. ■

For the maximization of Equation 1.13, we take into account two things: First, since $\theta_{ij} = \{\theta_{ijk}\}_{k=1}^{r_i}$ is a probabilistic distribution, then $\sum_{k=1}^{r_i} \theta_{ijk} = 1$. Second,

the parameters independence assumptions allows us to compute separately each parameter θ_{ij}, which leads us to:

$$\max_{\mathbf{\Theta}} P(\mathcal{D}|\mathcal{G},\mathbf{\Theta}) = \prod_{i=1}^{n}\prod_{j=1}^{q_i}\underbrace{\left[\max_{\theta_{ij}}\prod_{k=1}^{r_i}(\theta_{ijk})^{N_{ijk}}\right]}_{\sum_k \theta_{ijk}=1} \tag{1.14}$$

Letting $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$, be the number of times that the parents of X_i take the values $\mathbf{pa}(X_i)^{(j)}$ in $\mathcal{D}$. In Equation 1.14 we see that the expression inside brackets satisfies the conditions of Theorem 1.5.1, assuming N_{ij} is positive. So the value Θ that maximizes the likelihood can be described simply as:

$$\theta_{ijk}^{MLE} = \frac{N_{ijk}}{N_{ij}} \tag{1.15}$$

1.5.7 Bayesian estimation of parameters

MLE has two caveats. First and more obvious, the requirement that N_{ij} to be always positive. This is often not the case, especially in small datasets $\mathcal{D}$ or when the number of variables n is big. Second, it does not allow us to use the *a priori* information of Θ we might have. These two caveats can be solved treating $\mathcal{D}$ and $\mathbf{\Theta}$ both as random variables, and assuming that there exists a prior distribution $\pi(\mathbf{\Theta}|\mathcal{G}) = \sum_{\mathcal{D}} P(\mathbf{\Theta},\mathcal{D}|\mathcal{G})$, over all possible values of $\mathcal{D}$. That prior represents more an initial (often subjective) belief than a probability, and that belief is meant to be updated with the arrival of an actual value of $\mathcal{D}$ by applying Bayes theorem to the likelihood function:

$$P(\mathbf{\Theta}|\mathcal{G},\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{G},\mathbf{\Theta})\pi(\mathbf{\Theta}|\mathcal{G})}{P(\mathcal{D}|\mathcal{G})} \propto P(\mathcal{D}|\mathcal{G},\mathbf{\Theta})\pi(\mathbf{\Theta}|\mathcal{G}) \tag{1.16}$$

In this equation $P(\mathcal{D}|\mathcal{G})$ can be omitted since it does not affect the value of $\mathbf{\Theta}$ when maximizing the posterior distribution $P(\mathbf{\Theta}|\mathcal{G},\mathcal{D})$ (recall that, when we estimate the parameters, $\mathcal{G}$ is known). As in Equation 1.16, Bayesian estimation has typically the form $posterior \propto likelihood \times prior$, where the posterior represents the prior (old belief) confronted with evidence. Since that belief update is susceptible to be made more than once, it is very convenient that the prior and the posterior are both of the same family of probability distributions, which gives rise to the following concept:

Definition 1.5.1 (Conjugate distribution, Conjugate prior)
[Schlaifer and Raiffa, 1961] If the posterior and prior distributions are in the same probability distribution family, they are said to be conjugate distributions, and the prior is called a conjugate prior for the likelihood function.

From Equation 1.13 we realize that the probability of θ_{ij} follows a multinomial distribution:

$$P(\theta_{ij}|\mathcal{G},\mathcal{D}) = \prod_{k=1}^{r_i} (\theta_{ijk})^{N_{ijk}}$$

Being the Dirichlet distribution the conjugate prior of the multinomial distribution (see Definition 1.5.1), we choose it to model the prior and the posterior distributions:

$$\begin{aligned} \pi(\mathbf{\Theta}|\mathcal{G}) \quad &\propto \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i} (\theta_{ijk})^{\alpha_{ijk}} \\ P(\mathbf{\Theta}|\mathcal{G},\mathcal{D}) &\propto \prod_{i=1}^{n}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i} (\theta_{ijk})^{N_{ijk}+\alpha_{ijk}} \end{aligned} \tag{1.17}$$

Where the parameters $\alpha_{ijk} > -1$. Equation 1.17 can be optimized using Theorem 1.5.1, as we did for Equation 1.15:

$$\theta_{ijk}^{MAP} = \frac{N_{ijk} + \alpha_{ijk}}{N_{ij} + \alpha_{ij}} \tag{1.18}$$

Where MAP stands for *maximum a posteriori*, and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$.

1.5.8 Structure Learning

We mentioned that BN structure learning (BNSL) can be done by using constraint-based approaches, score-based approaches and hybrid approaches; in this thesis we will focus on score-based approaches, which typically make use of a scoring function to guide a local search until a local maximum is achieved. In that context, we mentioned as well that there are different possible search spaces: the DAG search space, the Markov equivalence class search space and the variable ordering search space, in this work we will discuss only about the first one.

Log-likelihood score: From the score-based approach, BNSL is a model selection, where each candidate graph $\mathcal{G}$ we evaluate generates a (unique) set of parameters $\mathbf{\Theta}^{\mathcal{G}}$ for a given learning dataset $\mathcal{D}$. A natural way to evaluate how well the model $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ fits the data $\mathcal{D}$ is to evaluate the logarithm (in base 2) of its likelihood $\log(P_{\mathcal{B}}(\mathcal{D}))$ (aka. log-likelihood); this is equivalent to taking the logarithm of Equation 1.13, which is presented as a score function in 1.5.2. This has many interesting properties: First, from an information-theoretic point of view, $-\log P_{\mathcal{B}}(\mathcal{D})$ represents the expected number of bits required to optimally encode $\mathcal{D}$ [Cover and Thomas, 2006], so that the log-likelihood expresses how well $\mathcal{B}$

compresses the learning data $\mathcal{D}$. Second, the log-likelihood score is decomposable (see Definition 1.5.3), *i.e.* modifications into $\mathcal{G}$ require recomputing the score only in the concerned nodes, which is very handy to perform quick computations when performing a BNSL. Third, the log-likelihood is score-equivalent (see Definition 1.5.4), which is a desirable property since Markov-equivalent models encode exactly the same independences. In addition, the log-likelihood does not add any set of hyper-parameters to calibrate, which makes it easy to apply.

Definition 1.5.2 (Log-likelihood score)

$$Score_{LL}(\mathcal{G}, \mathcal{D}) = \log\left(P(\mathcal{D}|\mathcal{G}, \mathbf{\Theta}^{\mathcal{G}})\right) = \sum_{i=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{r_i} N_{ijk} \log \theta_{ijk}^{\mathcal{G}} \tag{1.19}$$

Definition 1.5.3 (Score decomposition) *A scoring function is called decomposable iff it can be expressed as the sum of local scores for every node conditioned on its parents:*

$$Score(\mathcal{G}, \mathcal{D}) = \sum_{i=1}^{n} Score(X_i|\mathbf{Pa}(X_i), \mathcal{D})$$

Definition 1.5.4 (Score equivalence) *A scoring function is called equivalent if it assigns the same score to any member of a given Markov equivalence class.*

However, in practice we do not use the log-likelihood score because it tends to over-fit: there is no score penalization choosing a candidate graph $\mathcal{G}$ with many edges (or even complete), so that a more complex model tends to give a higher score w.r.t. $\mathcal{D}$. To solve that particular problem, the AIC and BIC scores are introduced.

AIC score: [Akaike, 1970] Having a candidate model $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta}^{\mathcal{G}})$ to approximate the (unknown) probability distribution $P(\cdot)$ that generated the learning dataset $\mathcal{D}$, the approximation quality of $\mathcal{B}$ can be measured by its Kullback-Leibler divergence w.r.t. the true distribution P (see definition 1.5.5)

Definition 1.5.5 (Kullback-Leibler Divergence) *[Kullback and Leibler, 1951] The Kullback-Leibler divergence (aka. relative entropy) is a measure of how close two probabilistic distributions (or densities) P and Q are.*

$$D_{KL}(P\|Q) = \mathbb{E}_P\left[\log \frac{P}{Q}\right]$$

Its value is zero iff P and Q are identical. In the typical case, it measures how well an estimated distribution Q approximates a true underlying distribution P, being meant to be minimized.

$$D_{KL}(P(\mathbf{X})\|P_{\mathcal{B}}(\mathbf{X})) = \sum_{\mathbf{x}\in\Omega_{\mathbf{X}}} P(\mathbf{x}) \log \frac{P(\mathbf{x})}{P_{\mathcal{B}}(\mathbf{x})}$$
$$= \sum_{\mathbf{x}\in\Omega_{\mathbf{X}}} P(\mathbf{x}) \log P(\mathbf{x}) - \sum_{\mathbf{x}\in\Omega_{\mathbf{X}}} P(\mathbf{x}) \log P_{\mathcal{B}}(\mathbf{x})$$

In the preceding equation, since the first term does not depend of $\mathcal{B}$, the minimization of the Kullback-Leibler divergence is equivalent to the maximizing the following expression:

$$K = \sum_{\mathbf{x}\in\Omega_{\mathbf{X}}} P(\mathbf{x}) \log P_{\mathcal{B}}(\mathbf{x})$$

Needing to compute K, knowing $\mathcal{D} = \{\mathbf{x}^m\}_{m=1}^N$ we might intuitively think that a good estimate to compute it would be:

$$\overline{K} = \frac{1}{N} \sum_{m=1}^{N} \log P_{\mathcal{B}}(\mathbf{x}^m) = \frac{1}{N} Score_{LL}(\mathcal{G}, \mathcal{D})$$

However, $\overline{K}$ is a biased estimation of K because the dataset $\mathcal{D}$ was used twice: firstly to compute the parameters $\mathbf{\Theta}$ and then to compute the above estimation. In [Akaike, 1970], it was showed that this bias can be approximated as:

$$K - \overline{K} \approx \frac{\dim\left(\mathbf{\Theta}^{\mathcal{G}}\right)}{N}$$

Where $\dim\left(\mathbf{\Theta}^{\mathcal{G}}\right)$ represents the number of independent parameters in $\mathbf{\Theta}^{\mathcal{G}}$. Since $\mathbf{\Theta}^{\mathcal{G}} = \{\theta_i\}_{i=1}^n$, we can decompose its dimension as $\dim(\mathbf{\Theta}^{\mathcal{G}}) = \sum_{i=1}^n \dim(\theta_i)$. For any of the q_i values of j, θ_{ij} can take exactly r_i different values; but considering that $\sum_{k=1}^{r_i} P(X_i) = 1$, it is enough to take into account only $r_i - 1$ values for describing Θ_{ij} (*i.e.* there are only $r_i - 1$ independent values). Then, we can conclude that:

$$\dim \mathbf{\Theta}^{\mathcal{G}} = \sum_{i=0}^{n} (r_i - 1) q_i \tag{1.20}$$

Now we have everything set up to define the score AIC:

Definition 1.5.6 (Score AIC) *[Akaike, 1970] The Akaike Information Criterion (AIC) score is defined as follows:*

$$Score_{AIC}(\mathcal{G}, \mathcal{D}) = Score_{LL}(\mathcal{G}, \mathcal{D}) - \dim\left(\mathbf{\Theta}^{\mathcal{G}}\right)$$

MDL/BIC score: The MDL (minimum description length) principle [Lam and Bacchus, 1994] is a model selection criterion following the Occam's razor principle (the *simpler model* tends to be the right one). In the context of selecting probabilistic models, the MDL principle formalizes *simpler model* as the one that can describe the observed data $\mathcal{D}$ and itself with the minimum possible amount of bits (being that amount of bits the so-called description length). We decompose the description length as the sum:

$$DL = DL_{\mathcal{D}} + DL_{\mathcal{B}} \tag{1.21}$$

Where DL is the total description length (to minimize), which is the sum of the number of bits needed to encode the data $DL_{\mathcal{D}}$, plus the number of bits needed to encode the model $DL_{\mathcal{B}}$. Evidently, in our case the model is a BN, $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$.

The number of bits to encode $DL_{\mathcal{D}}$ is equal to $-\log(P_{\mathcal{B}}(\mathcal{D}))$ [Cover and Thomas, 2006], which corresponds to $-Score_{LL}(\mathcal{G}, \mathcal{D})$. The computation of $DL_{\mathcal{B}}$ is decomposed as $DL_{\mathcal{G}} + DL_{\mathbf{\Theta}}$. We need $-\log d$ bits to encode any integer number d (we treat indices i as integer numbers when encoding). As a consequence, to compute $DL_{\mathcal{G}}$ we specify the number of random variables: $-\log(n)$; their domain sizes: $-\sum_{i=1}^{n} \log r_i$; the number of parents and their identity for each node: $\log(n) \sum_{i=1}^{n} (1 + q_i)$. Values θ_{ijk}, being real numbers, can have different numbers of bits in their representations (depending on the precision used to represent them), we assume they can be encoded in $\log N/2$ bits each, because it is the usual value in the literature [Heckerman, 1995], then the computation of DL_{Θ} becomes simply $\log N \dim \mathbf{\Theta}/2$. Putting all together:

$$\begin{aligned} DL = &- \sum_{i=1}^{N} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \theta_{ijk} + \frac{\log N}{2} \dim \mathbf{\Theta} \\ &- \log(n) \sum_{i=1}^{n} (1 + q_i) - \log(n) - \sum_{i=1}^{n} \log r_i \end{aligned}$$

We notice that the last two terms of the preceding expression are constants w.r.t any possible candidate model $\mathcal{B}$ we could propose (neither the number of variables nor their domain sizes would change); then, we do not take them into account. Assuming that $N \gg n$ and looking at Equation 1.20 we realize that the third term of the preceding expression is negligible w.r.t the second one, in consequence we do not take it into account either. We reflect this considerations in the following equation:

$$DL' = - \sum_{i=1}^{N} \sum_{j=0}^{q_i} \sum_{k=0}^{r_i} N_{ijk} \log \theta_{ijk} + \frac{\log N}{2} \dim \mathbf{\Theta}$$

In order to describe a score function from that minimization problem, we take the negative value of DL':

Definition 1.5.7 (MDL Score) *[Lam and Bacchus, 1994] The minimum description length (MDL) score is defined as follows:*

$$\begin{aligned} Score_{MDL}(\mathcal{G}, \mathcal{D}) &= \sum_{i=1}^{N} \sum_{j=0}^{q_i} \sum_{k=0}^{r_i} N_{ijk} \log \theta_{ijk} - \frac{\log N}{2} \dim \mathbf{\Theta} \\ &= Score_{LL}(\mathcal{G}, \mathcal{D}) - \frac{\log N}{2} \dim \mathbf{\Theta} \end{aligned}$$

The MDL score can also be derived using a Stirling approximation of the Bayesian measure $P(\mathcal{G}|\mathcal{D})$, this was done in [Schwarz, 1978]:

Definition 1.5.8 (BIC Score) *[Schwarz, 1978] The Bayesian information criterion (BIC) score is defined as follows:*

$$Score_{BIC}(\mathcal{G}, \mathcal{D}) = Score_{MDL}(\mathcal{G}, \mathcal{D})$$

Good properties of AIC and BIC: The AIC and BIC scores are essentially the log-likelihood score minus a penalization term for the complexity of $\mathcal{G}$. The penalization is bigger for the BIC score, so that it leads to learning structures with fewer edges than AIC. Since $\dim \mathbf{\Theta}$ is decomposable, and is the same for all members of a Markov equivalence class, then AIC and BIC have the properties of decomposition and equivalence (see Definitions 1.5.3 and 1.5.4). Assuming a big dataset $\mathcal{D}$ ($N \to \infty$), and that $\mathcal{D}$ was generated by a BN $\mathcal{B}^* = (\mathcal{G}^*, \mathbf{\Theta}^*)$, the log-likelihood score tends learn a model $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ that over-fits $\mathcal{D}$, because there is no penalization term, so adding any edge to $\mathcal{G}$ that is not in $\mathcal{G}^*$ will not decrease the score; moreover, due to statistical fluctuations in $\mathcal{D}$, it can actually (slightly) increase it. In the same scenario, if $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$ is the graph learned using the BIC or AIC scores, adding an edge to $\mathcal{G}$ not present in $\mathcal{G}^*$, the term $\dim(\mathbf{\Theta})$ is increased, then the penalization term grows and the score decreases. This desirable behavior of the BIC and AIC scores is formalized in the following concept:

Definition 1.5.9 (Score asymptotic consistency) *A score function is said to be asymptotically consistent if for any big dataset $\mathcal{D}$ ($N \to \infty$) generated following a model $\mathcal{B}^* = (\mathcal{G}^*, \mathbf{\Theta})$, it takes its maximal value in $\mathcal{G}^*$ (or one member of its Markov equivalence class):*

$$\mathcal{G}^* = \underset{\mathcal{G}}{Argmax} \left[Score(\mathcal{G}, \mathcal{D}) \right]$$

BD Score: [Heckerman et al., 1995] The Bayesian Dirichlet (BD) score is a score function that tries to approximate the so-called Bayesian score, using a Dirichlet prior distribution:

Definition 1.5.10 (Bayesian Score) *[Cooper and Herskovits, 1992b] In the context of BNLS, the Bayesian score is represented as posterior probability $P(\mathcal{G}|\mathcal{D})$:*

$$P(\mathcal{G}|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{G})\pi(\mathcal{G})}{P(\mathcal{D})}$$

Where $P(\mathcal{D}|\mathcal{G})$ is a likelihood measure, $\pi(\mathcal{G})$ is the prior distribution over the structure (we assume it to be constant for most of the cases, so we do not take it into account for the score), and $P(\mathcal{D}) = \sum_{\mathcal{G}'} P(\mathcal{D}|\mathcal{G}')P(\mathcal{G}')$ is a normalization constant that allows the posterior distribution to actually represent a probability distribution (being a constant, it is not a function of $\mathcal{G}$, so it is not taken into account either). Then we can define the Bayesian score as:

$$Score(\mathcal{G}, \mathcal{D}) \propto P(\mathcal{D}|\mathcal{G}) = \int_{\boldsymbol{\Theta}} P(\mathcal{D}|\mathcal{G}, \boldsymbol{\Theta})\pi(\boldsymbol{\Theta}|\mathcal{G})d\boldsymbol{\Theta} \tag{1.22}$$

The Bayesian score can be interpreted as the average likelihood for all possible values Θ, not being biased towards a single value like $\Theta^{\mathcal{G}}$ (*e.g.* the MLE estimation, see Equation 1.15), this intuitively tells us that this score is not prone to benefit over-fitting instances of $\mathcal{G}$. Although it can be very challenging to compute the value of the integral in Equation 1.22, it has a close form formula whenever its prior parameters follow a Dirichlet distribution $\pi(\theta_{ij}) \sim Dirichlet(\alpha_{ij1}, \ldots, \alpha_{ijr_i})$. To remain consistent with the definitions of the previous scores, and to avoid numerical issues when computing, we set the Bayesian scores not to express Equation 1.22, but its logarithm instead.

Definition 1.5.11 (BD Score) *[Heckerman et al., 1995] Being α_{ijk} Dirichlet prior hyper-parameters of $\pi(\boldsymbol{\Theta}|\mathcal{G})$, and $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$, then the BD score is defined as:*

$$Score_{BD}(\mathcal{G}, \mathcal{D}) = \log\left(\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij}+\alpha_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N_{ijk}+\alpha_{ijk})}{\Gamma(\alpha_{ijk})}\right) \tag{1.23}$$

Where $\Gamma(\cdot)$ is the gamma function, which is defined as $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}dx$, and $\log(\cdot)$ is the logarithm function in base 2.

K2 score: The BD requires to manually specify parameters α_{ijk}, having the caveat that they can be very numerous (one for every value θ_{ijk}). A practical solution to this problem is to use a non-informative prior distribution, like the Jeffreys' prior [Jeffreys, 1946] (which for the Dirichlet distribution proposes to set all the hyper-parameters to $1/2$). Following this logic, [Cooper and Herskovits, 1992b] proposed the K2 score, where all the hyper-parameters are set to one.

Definition 1.5.12 (K2 score) *[Cooper and Herskovits, 1992b] The K2 score is the particular case of the BD score where all parameters $\alpha_{ijk} = 1$. Knowing that $\Gamma(n+1) = n!$ for any non-negative integer n, and replacing the values α_{ijk} in Equation 1.23, we define:*

$$Score_{K2}(\mathcal{G}, \mathcal{D}) = \log\left(\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}N_{ijk}!\right) \tag{1.24}$$

BDe score: Another option for specifying the prior parameters for the BD score is to assume they follow the distribution of a prior model $\mathcal{B}_0$ and that this model has been learnt by a dataset with N' (imaginary) records (which we call pseudo-counts). We use that idea to define the Bayesian Dirichlet equivalent (BDe) score:

Definition 1.5.13 (BDe score) *[Heckerman et al., 1995] The BDe score is defined as the BD score (see Equation 1.23) where the prior parameters are specified as follows:*

$$\alpha_{ijk} = N' P(X_i = x_k, \mathbf{Pa}(X_i) = \mathbf{pa}(X_i)^{(j)} | \mathcal{B}_0) = N'_{ijk}$$

Where $\mathcal{B}_0$ is a BN defined over the same variables as $\mathcal{G}$, and N' is a positive number that represents the pseudo-counts (which can be interpreted as a weight of the prior factor).

$$Score_{BDe}(\mathcal{G}, \mathcal{D}) = \log\left(\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(N'_{ij})}{\Gamma(N_{ij}+N'_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N_{ijk}+N'_{ijk})}{\Gamma(N'_{ijk})}\right) \tag{1.25}$$

Where $N'_{ij} = \sum_{k=1}^{r_i} N'_{ijk}$.

Notice that $\mathbf{Pa}(X_i)$ do not represent the parents in $\mathcal{B}_0$ but the ones in $\mathcal{G}$ instead. The BD score, at least in the general case, has not the score equivalence property. In fact, [Heckerman et al., 1995] showed that the BDe score is the unique particular case of BD where it is score equivalent (clearly not being the score K2 in this case).

BDeu score: The main caveat of using directly the BD score is to find a suitable $\mathcal{B}_0$ (which often requires expert knowledge). For this reason [Buntine, 1991] proposed to set the parameters of the BDe score imposing a uniform prior (equivalent to having a empty graph for $\mathcal{B}_0$) where $P(X_i = x_k) = 1/q_i$, and having as only free parameter the number of pseudo-counts $N^{'}$, so that $N^{'}_{ijk} = N^{'}/(r_i q_i)$, which leads us to $N^{'}_{ij} = \sum_{k=1}^{q_i} N^{'}_{ijk} = N^{'}/q_i$.

Definition 1.5.14 (BDeu Score) *[Buntine, 1991] The Bayesian Dirichlet equivalent uniform (BDeu) score is defined as follows:*

$$Score_{BDeu}(\mathcal{G}, \mathcal{D}) = \log\left(\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\frac{N^{'}}{q_i})}{\Gamma(N_{ij}+\frac{N^{'}}{q_i})}\prod_{k=1}^{r_i}\frac{\Gamma(N_{ijk}+\frac{N^{'}}{r_i q_i})}{\Gamma(\frac{N^{'}}{r_i q_i})}\right) \tag{1.26}$$

Where $N^{'}$ is a positive number, and is the only free parameter to compute the score BDeu.

BDs score: The BDeu score is well-known to be very sensitive to the choice of its only parameter, [Silander et al., 2007] made an empiric study of the influence of the pseudo-counts, which delivers a counter-intuitive conclusion: bigger values of $N^{'}$ lead to models with more edges (this is unexpected because the $\mathcal{B}_0$ used for the BDeu prior is one with no edges). [Suzuki, 2017] states that for finite (although big) values of N, the BDeu score might violate what he calls the *regularity*[16] property; [Scutari, 2017] explained in more details the same phenomenon, stating a theorem saying that it will happen whenever there are some instantiations of $\mathbf{Pa}(X_i)$ missing in $\mathcal{D}$[17]. To overcome this issue, [Scutari, 2016] presents the BDs score, which is based on the BDeu score replacing values q_i with the number of different instantiations of the parents of X_i present in $\mathcal{D}$:

Definition 1.5.15 (BDs Score) *[Scutari, 2016] The Bayesian Dirichlet sparse (BDs) score is defined as:*

$$Score_{BDs}(\mathcal{G}, \mathcal{D}) = \log\left(\prod_{i=1}^{n}\prod_{j=1}^{\tilde{q}_i}\frac{\Gamma(\frac{N^{'}}{\tilde{q}_i})}{\Gamma(N_{ij}+\frac{N^{'}}{\tilde{q}_i})}\prod_{k=1}^{r_i}\frac{\Gamma(N_{ijk}+\frac{N^{'}}{r_i \tilde{q}_i})}{\Gamma(\frac{N^{'}}{r_i \tilde{q}_i})}\right) \tag{1.27}$$

Where $\tilde{q}_i$ is the number of different instantiations of $\mathbf{Pa}(X_i)$ present in $\mathcal{D}$.

[16] We (informally) define the *regularity* property of a score function: having models $\mathcal{G}$ and $\mathcal{G}^{'}$, if $\mathcal{G}$ is less complex than $\mathcal{G}^{'}$ and at the same time it has a better likelihood w.r.t. $\mathcal{D}$, implies that $Score(\mathcal{G}, \mathcal{D}) > Score(\mathcal{G}^{'}, \mathcal{D})$; then the score function is said to be regular.

[17] [Scutari, 2017] actually expresses it in terms of the *maximum entropy principle* [Shore and Johnson, 1980] rather than regularity.

The BDs score is not as sensitive to the choice of the parameter N' as BDeu, in addition it respects the regularity property proposed in [Suzuki, 2017], and the maximum entropy principle [Shore and Johnson, 1980]; however, to do so, it sacrifices the score equivalence property, being only asymptotically equivalent (when $N \to \infty$). The score equivalence property, while desirable, in practice is not a crucial property except when we want $\mathcal{G}$ to express causal relations [Pearl, 2009].

NML-based scores: All previously seen scores turn out to be very biased when the dataset size N is small. To overcome this issue, a regret term is defined:

$$\mathcal{R}(\bar{P}) = -\log \bar{P}(\mathcal{D}) - \inf_{P \in \mathcal{M}} [-\log P(\mathcal{D})] \tag{1.28}$$

From a MDL perspective, the regret $\mathcal{R}(\bar{P})$ represents the number of bits used to encode a dataset $\mathcal{D}$ using a hypothesis model $\bar{P}$, minus the minimum number of bits that is actually needed to describe that data (for a single element P of a given model class $\mathcal{M}$). We realize that different values of $\mathcal{D}$ result in different regret values; in order to learn a robust model, we consider the maximum (worst) regret value within the space $\mathcal{D}^N$ of all possible datasets with N records:

$$\mathcal{R}_{\max}(\bar{P}) = \sup_{\mathcal{D} \in \mathcal{D}^N} \left[-\log \bar{P}(\mathcal{D}) - \inf_{P \in \mathcal{M}} [-\log P(\mathcal{D})] \right] \tag{1.29}$$

Then, it is shown in [Shtar'kov, 1987] that the unique distribution solving the problem $P = \text{Argmin}_{\bar{P}} \mathcal{R}_{max}(\bar{P})$ is the NML (normalized maximum likelihood) distribution:

$$P_{NML}(\mathcal{D}|\mathcal{M}) = \frac{\hat{P}(\mathcal{D}|\mathcal{M})}{\sum_{\mathcal{D}' \in \mathcal{D}^N} \hat{P}(\mathcal{D}'|\mathcal{M})} \tag{1.30}$$

Where $\hat{P}(\mathcal{D}|\mathcal{M})$ is a MLE of P. We use NML as a score function in a BN context.

Definition 1.5.16 *The NML score is defined as follows:*

$$Score_{NML}(\mathcal{G}, \mathcal{D}) = \log(P_{NML}(\mathcal{D}|\mathcal{G})) = \log \left(\frac{P(\mathcal{D}|\Theta^{\mathcal{G}}(\mathcal{D}))}{\sum_{\mathcal{D}' \in \mathcal{D}^N} P(\mathcal{D}'|\Theta^{\mathcal{G}}(\mathcal{D}'))} \right) \tag{1.31}$$

Where $\Theta^{\mathcal{G}}(\cdot)$ are MLE parameters of a BN whose graph is $\mathcal{G}$.

The NML score is clearly score-equivalent, since every term of the form $P(\mathcal{D}'|\Theta^{\mathcal{G}})$ is the likelihood of $\mathcal{G}$ wrt. $\mathcal{D}'$, and the (log-)likelihood is a score equiv-

alent function. The logarithm of the denominator in Equation 1.31 is called *parametric complexity*[18]. Unfortunately, its computation is often unfeasible (notice that $\mathcal{D}'$ sums over all possible datasets with N records), and moreover its presence makes NML a non-decomposable score. However, [Silander et al., 2008] points out that for a single multinomial variable taking r different possible values, the parametric complexity can be computed as follows:

$$c_N^r = \sum_{k_1+\cdots+k_r=N} \frac{n!}{\prod_{j=1}^r k_j} \prod_{j=1}^r \left(\frac{k_j}{n}\right) \tag{1.32}$$

Which is shown in [Grünwald, 2007] to be computable in linear time in a recursive fashion. Moreover, [Silander et al., 2018] computes it using an approximation proposed by [Szpankowski and Weinberger, 2012]:

$$c_N^r \approx N\left(\log\alpha + (\alpha+2)\log C_\alpha - \frac{1}{C_\alpha}\right) - \frac{1}{2}\log\left(C_\alpha + \frac{2}{\alpha}\right) \tag{1.33}$$

Where $\alpha = \frac{r}{N}$ and $C_\alpha = \frac{1}{2} + \frac{1}{2}\sqrt{1+\frac{4}{\alpha}}$. Noticing that $P(X_i|\mathbf{Pa}_\mathcal{G}(X_i) = \mathbf{pa}_\mathcal{G}(X_i)^{(j)})$ follows a multinomial distribution, its NML can be computed as follows:

$$P_{NML}^1(\mathcal{D}_{ij}|\mathcal{G}) = \frac{P(\mathcal{D}|\hat{\theta}(\mathcal{D}_{ij},\mathcal{G}))}{\sum_{\mathcal{D}'} P(\mathcal{D}'|\hat{\theta}(\mathcal{D}'_{ij},\mathcal{G}))} \tag{1.34}$$

Where $\mathcal{D}_{ij}$ is the column of $\mathcal{D}$ corresponding to the random variables X_i, considering only the records where $\mathbf{Pa}_\mathcal{G}(X_i) = \mathbf{pa}_\mathcal{G}(X_i)^{(j)}$. $\mathcal{D}'$ iterates over all possible (one-dimensional) datasets with the form $\{x_i^{(m)}\}_{m=1}^{|\mathcal{D}_{ij}|}$ where $x_i^{(m)} \in \{x_1,\ldots,x_{r_i}\}$. $\hat{\Theta}(\cdot)$ are MLE local parameters wrt. a given dataset.

Definition 1.5.17 (fNML score) *[Silander et al., 2008] proposes to sacrifice the score-equivalence property of the NML score function in order to gain decomposability by using Equation 1.34 as a local score, computing its denominator in linear time using Equation 1.32 or approximating it with Equation 1.33, then defining the fNML (factorized NML) score:*

$$Score_{fNML}(\mathcal{G},\mathcal{D}) = \sum_{i=1}^n \sum_{j=1}^{q_i} \log P_{NML}^1(\mathcal{D}_{ij}|\mathcal{G}) \tag{1.35}$$

[18] In the literature it is also called *regret*, but it does not refer to the regret term presented in Equation 1.28.

Even if the fNML score is not score equivalent, it is proven in [Silander et al., 2008] that it behaves like the BIC score for big datasets, so it is asymptotically score equivalent; it has also the clear advantage of being non-parametric, while being almost as fast (or slow) to compute as the BDeu or BIC scores. Empirical analysis has shown, nonetheless, that the fNML score tends to prefer graphs with too many arcs even when it penalizes high complexity models [Silander et al., 2018].

In order to overcome the parsimony issues of the fNML and bring back the desirable score equivalence property, [Silander et al., 2018] derives a new score function:

Definition 1.5.18 (qNML Score) *[Silander et al., 2018] proposes to treat* $\{X_i\} \cup \mathbf{Pa}_{\mathcal{G}}(X_i)$ *and* $\mathbf{Pa}_{\mathcal{G}}(X_i)$ *as single multinomial variables (taking* $r_i q_i$ *and* q_i *different categorical values respectively), and computing the local score as the following quotient:*

$$P(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i)) = \frac{P(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i))}{P(\mathbf{Pa}_{\mathcal{G}}(X_i))}$$

Then the quotient NML (qNML) score is defined as follows:

$$Score_{qNML}(\mathcal{G}, \mathcal{D}) = \sum_{i=1}^{n} \frac{\log P^1_{NML}(\mathcal{D}_{\{X_i\}\cup\mathbf{Pa}_i}|\mathcal{G})}{P^1_{NML}(\mathcal{D}_{\mathbf{Pa}_i}|\mathcal{G})} \tag{1.36}$$

Where $\mathcal{D}_{\{X_i\}\cup\mathbf{Pa}_i}$ *and* $\mathcal{D}_{\mathbf{Pa}_i}$ *are one-dimensional arrays that come from* $\mathcal{D}$ *considering only the columns corresponding to the sets of variables* $\{X_i\} \cup \mathbf{Pa}_{\mathcal{G}}(X_i)$ *and* $\mathbf{Pa}_{\mathcal{G}}(X_i)$*, respectively.*

It is proved in [Silander et al., 2018] that the fNML and qNML respect the regularity condition present in [Suzuki, 2017]. The properties of all discussed scores are summarized in Table 1.1.

1.6 Dynamic Bayesian Networks

By definition, BNs represent static systems: they do not capture any temporal notion. Hence they are inadequate to cope with dynamic systems running over time. Dynamic Bayesian networks (DBN) [Dean and Kanazawa, 1989] have been precisely designed to take into account this temporal dimension. There are different flavors to DBNs, but the simplest one is certainly the 2-slice Temporal Bayesian Network (2TBN):

Definition 1.6.1 (Dynamic Bayesian network) *A 2TBN is a pair* $(\mathcal{B}_0, \mathcal{B}_{\rightarrow})$*, where:*

Score	Decompose.	Consistent	S. Equiv.	Regular	No Params.
LL	✓	✗	✓	✗	✓
AIC	✓	✓	✓	✓	✓
BIC	✓	✓	✓	✓	✓
BD	✓	✓	✗	✗	✗
K2	✓	✓	✗	✗	✓
BDe	✓	✓	✓	✗	✗
BDeu	✓	✓	✓	✗	✗
BDs	✓	✓	✓	✓	✗
NML	✗	✓	✓	✓	✓
fNML	✓	✓	✗	✓	✓
qNML	✓	✓	✓	✓	✓

Table 1.1: We describe the desired properties of all scores we have seen so far: Their decomposability (whether they can be expressed as the sum of local scores), their asymptotic consistency (whether they favor the true underlying distribution over other non-equivalents for large datasets), their score equivalence (whether they do no distinction between elements of the same Markov-equivalence class), their regularity (according to the definition of [Suzuki, 2017], and their absence of parameters (thus their easiness to be applied).

- *$\mathcal{B}_0 = (\mathcal{G}_0, \boldsymbol{\theta}_0)$ is a BN representing the uncertainty over the state of the system at time $t = 0$. Its set of random variables is $\mathbf{X}_0 = \{X_0^1, \ldots, X_0^n\}$ and, as a usual BN, $\mathcal{B}_0$ represents the joint probability distribution over $\mathbf{X}_0$.*

- *$\mathcal{B}_\rightarrow = (\mathcal{G}_\rightarrow, \boldsymbol{\theta}_\rightarrow)$ is a fragment of BN representing the transition of the state of the system from time t to time $t + 1$. Its graphical structure $\mathcal{G}_\rightarrow$ is such that:*

  - *its set of nodes is $\mathbf{X}_\rightarrow = \{X_t^1, \ldots, X_t^n, X_{t+1}^1, \ldots, X_{t+1}^n\}$. Nodes subscripted by t (resp. $t + 1$) correspond to random variables at time t (resp. $t + 1$).*
  - *there exists no arc between any pair of nodes (X_t^i, X_t^j), $i, j = 1, \ldots, n$, i.e., between any nodes at time t (but arcs between nodes at time $t + 1$ can exist).*
  - *there exists no arc from any node X_{t+1}^i to any node X_t^j, $i, j = 1, \ldots, n$, i.e., from a node at time $t + 1$ to a node at time t (backward-time arc).*

  Parameters $\boldsymbol{\theta}_\rightarrow$ are equal to $\{P(X_{t+1}^i | \mathbf{Pa}(X_{t+1}^i))\}_{i=1}^n$.

A grounded *DBN from time* $t = 0$ *up to time* $t = T$ *is a BN resulting from copy/pasting* $\mathcal{B}_0$ *and appending to it* $(T-1)$ *times fragment* $\mathcal{B}_{\rightarrow}$.

Figure 1.8 illustrates Definition 1.6.1: on the left side is displayed the 2TBN and, on the right part, a grounded DBN. As can be observed, the latter is obtained by copy/pasting once $\mathcal{B}_0$ and three times $\mathcal{B}_{\rightarrow}$. In Artificial Intelligence, DBNs defined by 2TBNs are a usual representation of uncertainty for dynamically evolving systems.

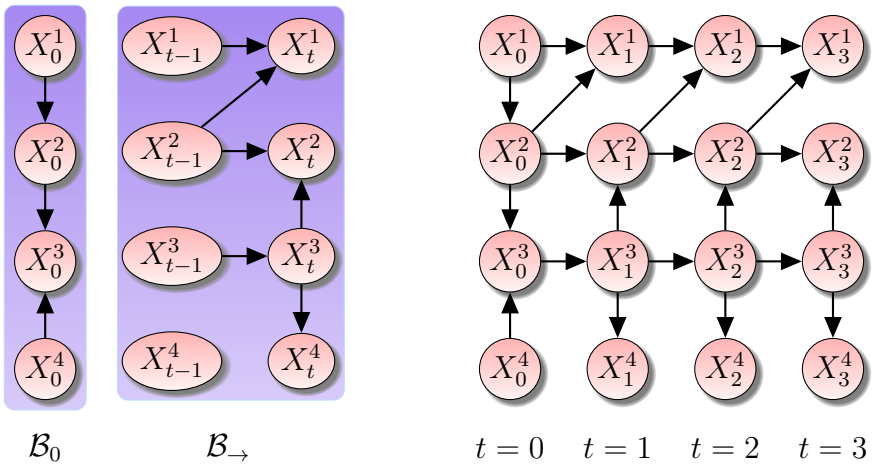


Figure 1.8: A DBN defined by its 2TBN (left) and one of its grounded DBN.

1.7 Non-Stationary DBNs

DBNs model dynamic systems, but those are supposed to have a certain stability, i.e., the transitions from time t to $t+1$ are supposed to remain the same over all time slices. In such cases, the transition probability distributions are said to be stationary. However, there are many situations in which such an assumption does not hold. Non-stationary DBNs (nsDBN) have been introduced precisely to cope with this kind of situations [Robinson and Hartemink, 2008, Robinson and Hartemink, 2010]. Essentially, they consist in modeling the dynamic system as a piecewise stationary model. In terms of DBNs, this amounts to model the system as a set of pairs (BN,time interval). Hence, similarly to classical DBNs, appending all these BNs on the time intervals on which they are defined results in a "grounded" BN defined over the union of all these intervals. More precisely:

Definition 1.7.1 *[Gonzales et al., 2015] A nsDBN is a sequence of pairs* $\langle(\mathcal{B}_h, T_h)\rangle_{h=0}^{m}$, *where* $T_0 = 0$ *represents the first time slice,* $\mathcal{B}_0$ *is the BN representing the distribution over the random variables at time 0, and each* $\mathcal{B}_h$, $h = 1, \ldots, m$, *is a BN*

representing the conditional probability of the random variables at time t given those at time $t-1$, for all t in time intervals $\mathbf{E}_h = \{T_{h-1}+1, \ldots, T_h\}$. T_h and $\mathbf{E}_h$ are called a transition time *and an* epoch *respectively. By convention, $\mathbf{E}_0 = \{0\}$.*

A DBN is a nsDBN in which the sequence contains only two pairs. Hence, The DBN of Figure 1.8 represents a nsDBN. Figure 1.9 shows another nsDBN which cannot be represented by a DBN because it involves more than two epochs [Gonzales et al., 2015]. Note that, by Definition 1.7.1, nsDBNs are defined over discrete time horizons.

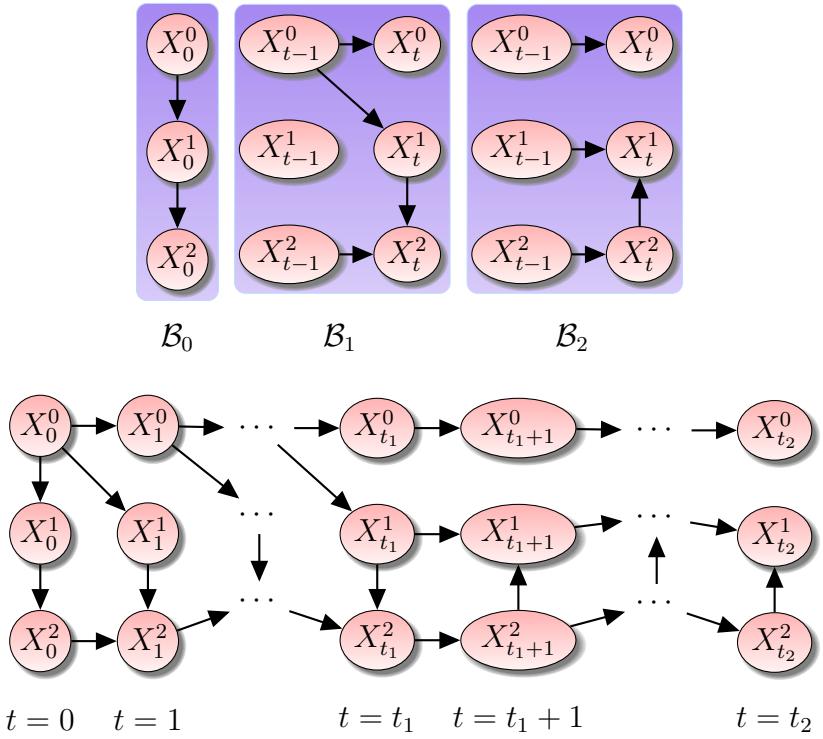


Figure 1.9: A nsDBN $\langle(\mathcal{B}_0, 0), (\mathcal{B}_1, t_1), (\mathcal{B}_2, t_2)\rangle$ (top of the figure) and its grounded BN (bottom), resulting from copying fragment $\mathcal{B}_0$ in the first time slice, copying fragment $\mathcal{B}_1$ in time slices 1 to t_1 and copying fragment $\mathcal{B}_2$ in time slices $t_1 + 1$ to t_2.

1.7.1 nsDBN learning issues

By definition, learning the structure and parameters of a 2TBN consists of applying separately classical learning algorithms on its two BNs: first, extract from Dataset $\mathcal{D}$ the subdatabase $\mathcal{D}_0$ whose columns correspond to the random variables

of time $t = 0$ and learn from $\mathcal{D}_0$ Bayesian network $\mathcal{B}_0$ using any algorithm presented in the above sections. For learning the parameters and structure of $\mathcal{B}_{\rightarrow}$, it is sufficient to reshape Dataset $\mathcal{D}$ into a new dataset $\mathcal{D}_{\rightarrow}$ in such a way that each row of $\mathcal{D}_{\rightarrow}$ corresponds to the random variables at times t and $t+1$, for some t. For instance, if the columns of $\mathcal{D}$ correspond to variables $\{\{X_i^t\}_{i=1}^n\}_{t=0}^T$, then each row of $\mathcal{D}$ is converted into T rows in $\mathcal{D}_{\rightarrow}$, the first one corresponding to Variables $\{X_i^0, X_i^1\}_{i=1}^n$, the second one to Variables $\{X_i^1, X_i^2\}_{i=1}^n$, and so on. Then, applying the learning algorithms of the preceding sections and enforcing that there exists no arc $X_i^{t+1} \rightarrow X_j^t$ nor any arc $X_i^t \rightarrow X_j^t$, it is possible to learn the structure and parameters of Bayesian network fragment $\mathcal{B}_{\rightarrow}$.

As a result, it is not more difficult to learn DBNs than to learn BNs. Learning general nsDBNs is more challenging because, in addition to learning the Bayesian network fragments $\mathcal{B}_h$ as described in the preceding paragraph, the algorithm also needs to determine the set of transition times T_h at which the network evolves. This explains why there exist very few algorithms in the literature to learn nsDBNs, see, e.g., [Nielsen and Nielsen, 2008, Grzegorczyk and Husmeier, 2009, Robinson and Hartemink, 2010]. The basic idea followed by [Robinson and Hartemink, 2010] is simply to adapt the above equations of the Bayesian Dirichlet learning framework to the case where there are several BN fragments in the model. For nsDBNs, the BD score can thus be adapted as:

$$P(\mathcal{G}_0, \ldots, \mathcal{G}_m) \prod_{h=0}^{m} \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij}(\mathbf{E}_h))}{\Gamma(N_{ij}(\mathbf{E}_h) + \alpha_{ij}(\mathbf{E}_h))} \times \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk}(\mathbf{E}_h) + \alpha_{ijk}(\mathbf{E}_h))}{\Gamma(\alpha_{ijk}(\mathbf{E}_h))},$$

where the $\mathcal{G}_h$'s are the structures at each epoch, and the quantities in the Gamma functions are similar to those in the BD score except that they are computed on subdatabases of $\mathcal{D}$ corresponding to epochs $\mathbf{E}_h$. Whenever the epochs are known, the learning amounts to perform the same operations as those done for 2TBNs, but if we do not know the epochs nor their number, the learning algorithm has to perform a search in the epochs space in addition to a search in the DAGs space, which is very time consuming and can lead to overfitting. [Robinson and Hartemink, 2010] provides such an algorithm but this one relies on optimization techniques that require the database to be defined over all the time slices over which the nsDBN will be exploited. This is not an issue in Robinson and Hartemink's paper because, in their target applications, the learning database is supposed to be a good representative of the subsequent situations in which the nsDBN will be exploited. However, their framework cannot be applied in situations where the model itself needs be updated over time when the nsDBN is exploited, e.g., when the nsDBN models uncertainties w.r.t. zero-day cyberattacks as fully new unexpected attacks may be created.

Learning the DBN structure in non-stationary contexts is therefore much more

challenging than learning in stationary contexts and people often have to make very strong assumptions to keep the search on the structures tractable. As an example, it is assumed in [Robinson and Hartemink, 2010] that the evolution of the structure over time follows a unique truncated exponential probability distribution. This assumption is questionable because it implies that removing an arc representing a strong dependence between two random variables is considered as equivalent to removing an arc that represents a very weak dependence. Other algorithms have been designed to learn nsDBNs, but they also make very strong assumptions. For instance, it is assumed in [Grzegorczyk and Husmeier, 2009] that only parameters can change. In [Nielsen and Nielsen, 2008], data are supposed to be constituted by only one observation per time slice, which requires that the generating process remains stationary for long periods. In [Dondelinger et al., 2010], random variables need be continuous, hence ruling out applications where data are discrete.

1.8 Conclusion

In this chapter, we reviewed the definition of BNs. We discussed about the inference procedures that can be performed through them as well as about the methods that can be used to learn their parameters from datasets. We also discussed about their use to describe processes over time.

We started by reviewing ground concepts of statistics such as probability distributions, random variables, conditional independence, Bayes theorem, etc.; we recalled some graph theory concepts as well. Then, we established the connection between graph theory and statistics in order to express joint multivariate distributions as products of terms associated to the nodes of a graph (which is called factorization). Based on that, we presented the concept of (discrete) BN, which is composed of a graph and a set of conditional probability tables (one for each node).

Then we studied the concept of inference over BNs, focusing our attention on the exact resolution of conditional probability queries: we studied in details the Variable Elimination and Shafer-Shenoy's algorithms. Also, we studied the problem of learning BNs from datasets. We pointed out the possible approaches to tackle this problem: Constraint-based, search-based, hybrid and exact approaches. We briefly discussed about the state-of-the-art literature of each one of those.

After that, we discussed about each of the most common assumptions made to learn BNs, which are: faithfulness, completeness, independence and identical

distribution, global/local independence, modularity and Dirichlet prior. Then we focused our attention on the search-based approaches, and in particular on the most widely used score functions to guide the search in the BN structure space. We discussed many score functions (K2, BIC, BDeu, fNML, etc.) and compared them based on their properties.

Finally, we discussed the introduction of the temporal dimension in BNs with the definition of DBNs, which describe stationary temporal processes. Also we discussed about ns-DBNs models, where we introduce the temporal notion of an *epoch*, so that the ns-DBNs are piecewise stationary processes within those, and we mentioned why the epoch notion makes the ns-DBNs particularly hard to learn.

Chapter 2

Bayesian Networks With Continuous Variables

In this chapter we discuss the usual ways to deal with the presence of continuous random variables in a BN context: First (and most obvious), it is possible to define appropriate discretization intervals [Friedman and Goldszmidt, 1996, Monti and Cooper, 1998, Mabrouk et al., 2015]. This has the advantage of producing discrete BNs (whose inference/learning methods are largely discussed in the literature). Second (and most widely used), we can exploit the Conditional Linear Gaussian models [Lauritzen and Wermuth, 1989], which deal directly with continuous variables and are well suited to perform fast computations. However, they lack expressive power, notably because they do not allow continuous nodes to be the parents of discrete nodes and because they are primarily defined to encode linear relationships. Third, it is possible to use mixtures of truncated basis function-based models [Moral et al., 2001, Shenoy, 2012, Langseth et al., 2012b], which are very expressive representations of mixed joint distribution. Unfortunately, their learning/inference are in practice too computationally expensive to scale up. We will briefly review these models in the rest of this chapter.

2.1 Discretization using BNs

From here on, to distinguish continuous random variables from discrete ones, we denote by $\mathring{X}_i$ a continuous variable and by X_i a discrete one. Without loss of generality, for any $\mathring{X}_i$, variable X_i with the same name but without the circle represents its discretized counterpart. Throughout the rest of this thesis, let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ and $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$ denote the set of discrete and continuous random variables respectively. We denote by $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$ the set of all random variables. Finally, for any variable X or set of random variables $\mathbf{Y}$

or $\mathring{\mathbf{Y}}$, let Ω_X (resp. $\Omega_{\mathbf{Y}}$ or $\Omega_{\mathring{\mathbf{Y}}}$) denote the domain of X (resp. $\mathbf{Y}$ or $\mathring{\mathbf{Y}}$). Following that notation, we define a variable discretization as follows:

Definition 2.1.1 (Discretization) *A discretization of a variable $\mathring{X}_i$ is a function $d_{\mathring{X}_i} : \Omega_{\mathring{X}_i} \rightarrow \{0, \ldots, g_i\}$ defined by an increasing sequence of g_i cutpoints $\{t_1, t_2, \ldots, t_{g_i}\} \subset \Omega_{\mathring{X}_i}$ such that:*

$$d_{\mathring{X}_i}(\mathring{x}_i) = \begin{cases} 0 & \text{if } \mathring{x}_i < t_1, \\ k & \text{if } t_k \leq \mathring{x}_i < t_{k+1}, \text{ for all } k \in \{1, \ldots, g_i - 1\} \\ g_i & \text{if } \mathring{x}_i \geq t_{g_i} \end{cases}$$

By abuse of notation we consider the discretization function of a discrete variable to be the identity function, i.e. $d_X(x) = x$.

Thus the discretized variable X_i corresponding to $\mathring{X}_i$ has a finite domain of $\{0, \ldots, g_i\}$ and, after discretizing all the continuous variables, the uncertainty over all the discrete and discretized variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ can be represented by a classical BN. This is achieved using a discretization policy:

Definition 2.1.2 (Discretization Policy) *A discretization policy is a set of discretization functions $\mathcal{F}_{\mathbf{X}}$, containing a discretization function $d_{\mathring{X}}$ for each $\mathring{X} \in \mathbf{X}$. When it is clear from the context, we call it simply $\mathcal{F}$.*

2.1.1 BN Learning and discretization

The formulation presented in this subsection comes from [Mabrouk et al., 2015]. Learning a discretized BN consists in finding the best pair $(\mathcal{G}, \mathcal{F})$ wrt. to a dataset $\mathring{\mathcal{D}}$ containing continuous variables:

$$(\mathcal{G}^*, \mathcal{F}^*) = \underset{(\mathcal{G},\mathcal{F})}{\text{Argmax}}\, P(\mathcal{G}, \mathcal{F}|\mathring{\mathcal{D}}) \tag{2.1}$$

By using the chain rule, we decompose the previous equation as the following product:

$$P(\mathcal{G}, \mathcal{F}|\mathring{\mathcal{D}}) = P(\mathcal{G}|\mathcal{F}, \mathring{\mathcal{D}})P(\mathcal{F}|\mathring{\mathcal{D}}) \tag{2.2}$$

Let $\mathcal{D}$ be the unique discretized database resulting from the discretization of $\mathring{\mathcal{D}}$ by discretization policy $\mathcal{F}$. Then $P(\mathcal{D}|\mathring{\mathcal{D}}, \mathcal{G}, \mathcal{F}) = 1$ and, by Bayes Theorem, the first product term $P(\mathcal{G}|\mathcal{F}, \mathring{\mathcal{D}})$ in Equation 2.2 is proportional to:

$$\begin{aligned} P(\mathcal{G}|\mathring{\mathcal{D}}, \mathcal{F}) \propto P(\mathring{\mathcal{D}}|\mathcal{G}, \mathcal{F}) &= \underbrace{P(\mathcal{D}|\mathring{\mathcal{D}}, \mathcal{G}, \mathcal{F})}_{equal\ to\ 1} P(\mathring{\mathcal{D}}|\mathcal{G}, \mathcal{F}) \\ &= P(\mathring{\mathcal{D}}|\mathcal{D}, \mathcal{G}, \mathcal{F})P(\mathcal{D}|\mathcal{G}, \mathcal{F}) \end{aligned} \tag{2.3}$$

Without additional information, it is not very restrictive to assume that all the databases $\mathring{\mathcal{D}}$ compatible with $\mathcal{D}$ given discretization policy $\mathcal{F}$ are equiprobable. In this case, $P(\mathring{\mathcal{D}}|\mathcal{D},\mathcal{G},\mathcal{F})$ is a constant and $P(\mathcal{G}|\mathring{\mathcal{D}},\mathcal{F}) \propto P(\mathcal{D}|\mathcal{G},\mathcal{F})$. As a result, determining the most likely structure given some *continuous* database is equivalent to maximizing the likelihood of its *discretized* counterpart, that is, maximizing:

$$P(\mathcal{G}|\mathring{\mathcal{D}},\mathcal{F}) \propto P(\mathcal{D}|\mathcal{G},\mathcal{F}) = \int_{\boldsymbol{\Theta}} P(\mathcal{D}|\mathcal{G},\mathcal{F},\boldsymbol{\Theta})\pi(\boldsymbol{\Theta}|\mathcal{F},\mathcal{G})\, d\boldsymbol{\Theta}. \tag{2.4}$$

Making the same assumptions done for learning BNs (completeness, faithfulness, iid. data, parameter independence, modularity and Dirichlet prior), Equation 2.4 can be solved using any approximation to the Bayesian score (see Definition 1.5.10) like the BD score (see Definition 1.5.11) or one of its particular cases. Note that, if the discretization policy $\mathcal{F}$ needs be determined, we also need a way to compute $P(\mathcal{F}|\mathring{\mathcal{D}})$ in Equation 2.2 in order to have a score function which can be used to explore the joint space of discretization policies and DAGs.

2.1.2 Monti & Cooper Discretization

[Monti and Cooper, 1998] proposes that $P(\mathcal{F}|\mathring{\mathcal{D}}) \propto P(\mathring{\mathcal{D}}|\mathcal{F})$ by assuming *a priori* that all the possible policies $\mathcal{F}$ are equiprobable, and assuming also that all discretization functions $d_{\mathring{X}} \in \mathcal{F}$ define the following uniform distributions:

$$P(\mathring{X} = \mathring{x}|x = k, d_{\mathring{X}}) = \frac{\mathbb{1}_{[t_k \leq \mathring{x} < t_{k+1}]}}{t_{k+1} - t_k} = \rho_k \tag{2.5}$$

Then, because of the iid. assumption in $\mathring{\mathcal{D}}$, we have that:

$$P(\mathring{\mathcal{D}}_{\mathring{X}}|d_{\mathring{X}}) = \prod_{k=1}^{r} \rho_k^{N_k} \tag{2.6}$$

where $\mathring{\mathcal{D}}_{\mathring{X}}$ is the column of Dataset $\mathring{\mathcal{D}}$ corresponding to Variable $\mathring{X}$. [Monti and Cooper, 1998] makes an additional independence assumption: the (uniform) density associated to each continuous variable depends only on its corresponding discretization (and not on the discretization of the other ones), which translates into:

$$P(\mathring{\mathcal{D}}|\mathcal{F}) = \prod_{\mathring{X} \in \mathring{\mathbf{X}}} P(\mathring{\mathcal{D}}_{\mathring{X}}|d_{\mathring{X}}) \tag{2.7}$$

Then we have all the elements required to define a score function:

Definition 2.1.3 (Monti & Cooper score) *The Monti & Cooper discretization score is defined as follows:*

$$Score_{MC}(\mathcal{F}, \mathcal{G}; \mathring{\mathcal{D}}) = \sum_{i=d+1}^{n} S_c(\mathring{X}_i, d_{\mathring{X}_i}; \mathring{\mathcal{D}}) + \sum_{i=1}^{n} S_d(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i), \mathcal{F}_i; \mathring{\mathcal{D}}) \quad (2.8)$$

Where $S_c(\cdot)$ is the score wrt. to the continuous variables, and is computed as the logarithm of Equation 2.6, $S_d(\cdot)$ corresponds to a discrete BN score, and can be computed as a local BD score (see Definition 1.5.11) and, $\mathcal{F}_i = d_{\mathring{X}_i} \cup \{d_{\mathring{X}} : \mathring{X} \in \mathbf{Pa}_{\mathcal{G}}(X_i)\}$. Thanks to Theorem 1.3.1 we know that changing the discretization $d_{\mathring{X}_i}$ would only influence the discretizations in the markov blanket of X_i; therefore, we can find a good discretization function by optimizing the following local score:

$$\begin{aligned} Score_{MC}(\mathring{X}_i, \mathcal{F}, \mathcal{G}; \mathring{\mathcal{D}}) = {} & S_c(\mathring{X}_i, d_{\mathring{X}_i}; \mathring{\mathcal{D}}) + S_d(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i), \mathcal{F}_i; \mathring{\mathcal{D}}) \\ & + \sum_{X_j \in \mathbf{Ch}_{\mathcal{G}}(X_i)} S_d(X_j, \mathbf{Pa}_{\mathcal{G}}(X_j), \mathcal{F}_j; \mathring{\mathcal{D}}) \end{aligned} \quad (2.9)$$

Unfortunately, [Monti and Cooper, 1998] does not provide a search algorithm over the space of discretizations.

2.1.3 Friedman's Discretization

Another attempt to learn BNs and their discretizations is proposed in [Friedman and Goldszmidt, 1996], where a MDL score approach is used (see Definition 1.5.7). In order to understand this approach, we introduce the following two information theory concepts: (conditional) entropy and mutual information (see Definitions 2.1.4 and 2.1.5).

Definition 2.1.4 (Entropy and Conditional Entropy) *Being $\mathcal{D}$ a dataset, and letting $\hat{P}_{\mathcal{D}}$ be a (frequence) probability measurement, the conditional entropy of $\mathbf{X}$ wrt. $\mathbf{Y}$ is defined as follows:*

$$H(\mathbf{X}|\mathbf{Y}) = -\sum_{\mathbf{x},\mathbf{y}} \hat{P}_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) \log \hat{P}_{\mathcal{D}}(\mathbf{x}|\mathbf{y})$$

Whenever $\mathbf{Y} = \emptyset$, we can denote it as $H(\mathbf{X})$ and call it simply entropy of $\mathbf{X}$.

Definition 2.1.5 (Mutual Information) *The mutual information of two random variable sets $\mathbf{X}$ and $\mathbf{Y}$ is defined as follows:*

$$I(\mathbf{X}, \mathbf{Y}) = \sum_{\mathbf{x},\mathbf{y}} \hat{P}_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) \log \frac{\hat{P}_{\mathcal{D}}(\mathbf{x}, \mathbf{y})}{\hat{P}_{\mathcal{D}}(\mathbf{x}) \hat{P}_{\mathcal{D}}(\mathbf{y})}$$

Since it can be easily shown that: $H(\mathbf{X}|\mathbf{Y}) = H(\mathbf{X}) - I(\mathbf{X}, \mathbf{Y})$, we can rewrite the term $DL_{\mathcal{D}}$ presented in Equation 1.21 as follows:

$$DL_{\mathcal{D}} = N \sum_{i=1}^{n} H(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i)) \tag{2.10}$$

$$DL_{\mathcal{D}} = N \sum_{i=1}^{n} H(X_i) - N \sum_{i=1}^{n} I(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i)) \tag{2.11}$$

$$DL_{\mathcal{D}} \leftarrow -N \sum_{i=1}^{n} I(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i)) \tag{2.12}$$

In Equation 2.12 we give an alternative definition of $DL_{\mathcal{D}}$ that ignores the terms $H(X)$, since they depend only of $\mathcal{D}$ and not on any parameters we intend to learn. The description length DL of Bayesian network $\mathcal{B}$ learnt from a dataset $\mathring{\mathcal{D}}$ (containing continuous variables), where a discretization policy $\mathcal{F}$ is applied, is defined as follows:

$$DL = DL_{\mathcal{B}} + DL_{\mathcal{D}} + DL_{\mathcal{F}} + DL_{X \to \mathring{X}} \tag{2.13}$$

Where $DL_{\mathcal{B}}$ is the number of bits required to store $\mathcal{B}$, $DL_{\mathcal{D}}$ is the number of bits required to store the discretized dataset $\mathcal{D}$, $DL_{\mathcal{F}}$ is the number of bits required to store all discretization policy parameters and $DL_{X \to \mathring{X}}$ represents the required information to encode the continuous values of $\mathring{\mathcal{D}}$ based on their discretized counterparts in $\mathcal{D}$.

We have that $DL_{\mathcal{B}}$ and $DL_{\mathcal{D}}$ are computed just like in the MDL score. On the other hand, for computing $DL_{\mathcal{F}}$ we consider that all potential cutpoints are midpoints in $\mathring{\mathcal{D}}$ (just like in Monti & Cooper discretization), and then we can assign an integer number for every possible combination of cutpoints, computing $DL_{\mathcal{F}}$ as:

$$DL_{\mathcal{F}} = \sum_{\mathring{X}_i \in \mathring{\mathbf{X}}_C} \log \binom{N_i - 1}{k_i - 1} \approx \sum_{\mathring{X}_i \in \mathring{\mathbf{X}}_C} (N_i - 1) H\left(\frac{k_i - 1}{N_i - 1}\right) \tag{2.14}$$

where $k_i = |\Omega_{X_i}|$ and $H(t) = -t \log(t) - (1-t) \log(1-t)$. The last formula results from the use of a Stirling's approximation in order to avoid computing the combinatorial terms. For computing $DL_{X \to \mathring{X}}$, we know that the description length of a single variable $\mathring{X}_i$ can be approximated as $-\log(\mathring{X}_i|X_i)$, thus:

$$DL_{X \to \mathring{X}} = -\sum_{\mathring{X} \in \mathring{\mathbf{X}}_C} \sum_{j=1}^{N} \log \hat{P}_{\mathring{\mathcal{D}}}(\mathring{x}_i^{(j)}|x_i^j) = N \sum_{\mathring{X} \in \mathring{\mathbf{X}}_C} H(\mathring{X}|X) \tag{2.15}$$

The preceding expression can be expressed in a more convenient way by means of the following proposition:

Proposition 2.1.1 ([Friedman and Goldszmidt, 1996]) *If $\mathcal{G}$ is a BN structure containing a variable X_i, which was discretized from a continuous variable $\mathring{X}_i$, using entropy and mutual information properties it can be shown that:*

$$H(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i)) + H(\mathring{X}_i|X_i) = H(\mathring{X}_i) - I(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i))$$

In Equation 2.13, we compute $DL_{\mathring{\mathcal{D}}}$ as in Equation 2.10, applying Proposition 2.1.1. Noticing once again that $H(\mathring{X}_i)$ depends neither on $\mathcal{G}$ nor on $\mathcal{F}$ (thus, it can be safely ignored for computing DL), we obtain that:

$$DL = DL_{\mathcal{B}} + DL_{\mathcal{F}} - N \sum_{i=1}^{n} I(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i)) \tag{2.16}$$

Definition 2.1.6 (Friedman & Goldsmith Score) *Seeing Equation 2.12, we can conclude that:*

$$DL = DL_{\mathcal{B}} + DL_{\mathcal{F}} + DL_{\mathcal{D}} \tag{2.17}$$

Then we have all the required elements for defining a score function:

$$\begin{aligned} Score_{FG}(\mathcal{F}, \mathcal{G}; \mathring{\mathcal{D}}) &= -DL_{\mathcal{B}} - DL_{\mathcal{D}} - DL_{\mathcal{F}} \\ &= Score_{MDL}(\mathcal{G}, \mathcal{D}) + \sum_{i:\mathring{X}_i \in \mathring{\mathbf{X}}_C} (N_i - 1) H\left(\frac{k_i - 1}{N_i - 1}\right) \end{aligned} \tag{2.18}$$

Knowing that the MDL score is decomposable, and that the discretization of a single variable will affect all its Markov blanket, a local score is also defined:

$$\begin{aligned} Score_{FG}(\mathring{X}_i, \mathcal{F}, \mathcal{G}; \mathring{\mathcal{D}}) = &-\frac{1}{2}\log N \left(q_i(r_i - 1) \; + \sum_{X_{i'} \in \mathbf{Ch}_{\mathcal{G}}(X_i)} q_{i'}(r_{i'} - 1) \right) \\ &- \log(k_i) + (N_i - 1) H\left(\frac{k_i - 1}{N_i - 1}\right) \\ &+ N \left[I(X_i, \mathbf{Pa}_{\mathcal{G}}(X_i)) \; + \sum_{X_{i'} \in \mathbf{Ch}_{\mathcal{G}}(X_i)} I(X_{i'}, \mathbf{Pa}_{\mathcal{G}}(X_{i'})) \right] \end{aligned} \tag{2.19}$$

In [Friedman and Goldszmidt, 1996] there is also a proposed algorithm for learning discretization policies and BNs at the same time in an iterative process:

From a given discretization $\mathcal{F}$, a discretized dataset $\mathcal{D}$ is obtained, from which a BN structure $\mathcal{G}$ is learned. From $\mathcal{G}$, an improved discretization policy $\mathcal{F}$ is computed, and this process is repeated in an iterative fashion (see details in Algorithm 2.1). The discretization of the variables is made one by one, knowing that if we re-discretize a continuous random variable, we must recompute the discretization of all its Markov Blanket (see Subroutine 2.2). When discretizing a single variable, we make the assumption that all the other variables are discrete (or already discretized), in order to add cutpoints in a greedy way until the local score is not improved anymore (see Subroutine 2.3).

Algorithm 2.1: BN multivariate discretization.

Input: $\mathring{\mathcal{D}}$: A dataset, $\mathcal{F}_0$: an initial discretization of $\mathring{\mathbf{X}}_C$
Output: $\mathcal{F}$: a locally optimal discretization of $\mathring{\mathbf{X}}_C$, $\mathcal{G}$: a BN structure.

$\mathcal{G}_0 \leftarrow$ empty graph; $\mathcal{G} \leftarrow$ empty graph; $\mathcal{F} \leftarrow \mathcal{F}_0$;
repeat
  $\mathcal{G}_0 \leftarrow \mathcal{G}$;
  $\mathcal{D} \leftarrow$ discretization of $\mathring{\mathcal{D}}$ using $\mathcal{F}$;
  $\mathcal{G} \leftarrow greedySearch(\mathcal{D})$;// Using MDL Score
  $\mathcal{F} = discretize(\mathcal{G}, \mathring{\mathcal{D}}, \mathcal{F}_0)$;
until $\mathcal{G} = \mathcal{G}_0$;

Subroutine 2.2: Multivariate Discretization.

discretize $(\mathcal{G}, \mathring{\mathcal{D}}, \mathcal{F}_0)$
  $Q \leftarrow$ queue with all $\mathring{X} \in \mathring{\mathbf{X}}_C$; $\mathcal{F} \leftarrow \mathcal{F}_0$;
  while Q *is not empty* **do**
    Remove first element $\mathring{X}$ from Q;
    $d'_{\mathring{X}} = discretizeOneVariable(\mathring{X}, \mathcal{G})$;
    $\mathcal{F}' \leftarrow \mathcal{F}$ replacing $d_{\mathring{X}}$ by $d'_{\mathring{X}}$;
    if $Score_{FG}(\mathring{X}, \mathcal{F}', \mathcal{G}; \mathring{\mathcal{D}}) > Score_{FG}(\mathring{X}, \mathcal{F}, \mathcal{G}; \mathring{\mathcal{D}})$ **then**
      $\mathcal{F} \leftarrow \mathcal{F}'$;
      for $Y \in \mathbf{MB}_{\mathcal{G}}(X)$ **do**
        if $\mathring{Y} \in \mathring{\mathbf{X}}_C$ **then**
          Push $\mathring{Y}$ into Q;
  return $\mathcal{F}$;

Subroutine 2.3: Discretization of a single variable.

```
discretizeOneVariable (X̊, G, D̊, F)
    d'_X̊ ← an empty discretization (no cutpoints);
    T ← list of midpoints of D̊_X̊;
    score_prec ← −∞; addCutpoints = True;
    while addCutpoints do
        score_max ← −∞; t_best ← null;
        for t ∈ T do
            d' ← d'_X̊ adding t to its cutpoints;
            F' ← F replacing its d_X̊ element by d';
            score ← Score_FG(X̊, F', G; D̊);
            if score − score_max > 0 then
                score_max ← score; t_best ← t;
        if score_max > score_prec then
            Remove t_best from T, and add it as cutpoint of d'_X̊;
            Score_prec ← Score_max;
        else
            addCutpoints = False;
    return d'_X̊;
```

2.2 Conditional Linear Gaussian BNs

In this section we describe the Conditional Linear Gaussian (CLG) distribution, and its use in hybrid BNs, *i.e.* BNs defined over discrete and continuous random variables. When dealing with BNs, we typically express CPDs in tables (CPTs), but nothing in the definition of BNs forbids us to use any other type of CPD. The model studied in this section makes use of Gaussian distributions for expressing CPDs involving continuous variables (see Definition 2.2.1), which is a very common continuous probability distribution.

Definition 2.2.1 (Multivariate/Univariable Gaussian distribution) *Being* $\mathring{\mathbf{X}}$ *a vector of continuous random variables, a multivariate Gaussian (or normal) distribution is defined as follows:*

$$p(\mathring{\mathbf{X}} = \mathring{\mathbf{x}}) = \frac{1}{(2\pi)^{n/2}|\Sigma_{\mathring{\mathbf{X}}}|^{1/2}} \exp\left[-\frac{1}{2}(\mathring{\mathbf{x}} - \boldsymbol{\mu}_{\mathring{\mathbf{X}}})^T \Sigma_{\mathring{\mathbf{X}}}^{-1}(\mathring{\mathbf{x}} - \boldsymbol{\mu}_{\mathring{\mathbf{X}}})\right]$$

Where $\mu_{\mathring{\mathbf{X}}}$ *is a mean vector, computed as the expectation* $\mathbb{E}[\mathring{\mathbf{X}}]$*; and* $\Sigma_{\mathring{\mathbf{X}}}$ *is a covariance matrix, computed as* $\mathbb{E}[(\mathring{\mathbf{X}} - \boldsymbol{\mu}_{\mathring{\mathbf{X}}})^T(\mathring{\mathbf{X}} - \boldsymbol{\mu}_{\mathring{\mathbf{X}}})]$*. In the case of a*

single variable $\mathring{X}$, the mean vector becomes the mean of a single variable $\mu_{\mathring{X}}$, the covariance matrix becomes a variance $\sigma^2_{\mathring{X}}$. Then, a single-variable Gaussian (normal) distribution is defined as follows:

$$p(\mathring{X} = \mathring{x}) = \frac{1}{\sqrt{2\pi}\sigma_{\mathring{X}}} \exp\left[-\frac{1}{2}\frac{(\mathring{x} - \mu_{\mathring{X}})^2}{\sigma^2_{\mathring{X}}}\right]$$

The Gaussian distribution is often denoted as $\mathcal{N}(\mathring{\mathbf{X}}|\mu_{\mathring{\mathbf{X}}}, \Sigma_{\mathring{\mathbf{X}}})$.

A continuous variable can often be expressed (or approximated) as a linear function of other continuous variables, the CPD of such a variable can be modeled as a Conditional Linear Gaussian (see Definition 2.2.2), which has very appealing properties: Any CLG can be represented as a multivariate Gaussian distribution (see Theorem 2.2.1) et *vice versa* (see Theorem 2.2.2). A direct consequence of those properties is that we can take any BN structure $\mathcal{G}$ defined over discrete and continuous variables (as long as no continuous variable has a discrete child in $\mathcal{G}$) and define a BN model (see Definition 2.2.3).

Definition 2.2.2 (Conditional Linear Gaussian CPD) *Let $\mathring{Y}$ be a continuous random variable and $\mathring{\mathbf{X}} = [\mathring{X}_1, \ldots, \mathring{X}_k]$ a vector of continuous random variables, we say that $\mathring{Y}$ is a linear Gaussian model of $\mathring{\mathbf{X}}$ if there exist parameters α, $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_k]$ and σ^2 such that:*

$$P(\mathring{Y}|\mathring{\mathbf{X}}) = \mathcal{N}(\alpha + \boldsymbol{\beta}^T\mathring{\mathbf{X}}, \sigma^2)$$

This definition means that $\mathring{Y}$ is expressed as a linear function of the variables $\mathring{\mathbf{X}}$:

$$\mathring{Y} = \alpha + \boldsymbol{\beta}^T\mathring{\mathbf{X}} + \epsilon$$

Where ϵ is a random variable with mean 0 and variance σ^2.

Theorem 2.2.1 (GLG to Gaussian) *[Wermuth, 1980] Any CLG $P(\mathring{Y}|\mathring{\mathbf{X}})$ has the following two properties, assuming that $\mathring{\mathbf{X}} \sim \mathcal{N}(\mathring{\mathbf{X}}|\boldsymbol{\mu}_{\mathring{\mathbf{X}}}, \Sigma_{\mathring{\mathbf{X}}})$:*

1. *$P(\mathring{Y})$ is a Gaussian distribution $\mathcal{N}(\mathring{Y}|\mu_{\mathring{Y}}, \sigma^2_{\mathring{Y}})$ with $\mu_{\mathring{Y}} = \alpha + \boldsymbol{\beta}^T\boldsymbol{\mu}_{\mathring{X}}$ and $\sigma^2_{\mathring{Y}} = \sigma^2 + \boldsymbol{\beta}^T\Sigma_{\mathring{\mathbf{X}}}\boldsymbol{\beta}$.*

2. *The joint distribution $P(\mathring{Y}, \mathring{\mathbf{X}})$ is a multivariate Gaussian distribution where the element of the covariance matrix corresponding to the variables $\mathring{Y}$ and $\mathring{X}_i$ is computed as $\sum_{j=1}^{k} \beta_j \Sigma_{\mathring{\mathbf{X}}}[i, j]$.*

Theorem 2.2.2 (Gaussian to CLG) *[Wermuth, 1980] Having the following multivariate Gaussian distribution:*

$$P(\mathring{\mathbf{X}}, \mathring{Y}) = \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_{\mathring{\mathbf{X}}} \\ \mu_{\mathring{Y}}\end{pmatrix}, \begin{bmatrix}\Sigma_{\mathring{\mathbf{X}}\mathring{\mathbf{X}}} & \Sigma_{\mathring{\mathbf{X}}\mathring{Y}} \\ \Sigma_{\mathring{Y}\mathring{\mathbf{X}}} & \Sigma_{\mathring{Y}\mathring{Y}}\end{bmatrix}\right)$$

It can be expressed as a CLG $P(\mathring{Y}|\mathring{\mathbf{X}}) = \mathcal{N}(\mathring{Y}|\alpha + \boldsymbol{\beta}^T\mathring{\mathbf{X}}, \sigma^2)$*, where:*

1. $\alpha = \mu_{\mathring{Y}} - \Sigma_{\mathring{Y}\mathring{\mathbf{Y}}}\Sigma^{-1}_{\mathring{\mathbf{X}}\mathring{\mathbf{X}}}\boldsymbol{\mu}_{\mathring{\mathbf{X}}}$
2. $\boldsymbol{\beta} = \Sigma^{-1}_{\mathring{\mathbf{X}}\mathring{\mathbf{X}}}\Sigma_{\mathring{Y}\mathring{\mathbf{X}}}$
3. $\sigma^2 = \Sigma_{\mathring{Y}\mathring{Y}} - \Sigma_{\mathring{Y}\mathring{\mathbf{X}}}\Sigma^{-1}_{\mathring{\mathbf{X}}\mathring{\mathbf{X}}}\Sigma_{\mathring{\mathbf{X}}\mathring{Y}}$

Definition 2.2.3 (Conditional Linear Gaussian BN) *[Lauritzen and Wermuth, 1989] The CLG-BN model* $\mathcal{B} = (\mathcal{G}, \boldsymbol{\Theta})$ *represents a mixed probability distribution, i.e. a distribution with discrete and continuous variables. It is composed by a BN structure* $\mathcal{G}$ *defined over a (mixed) set of variables* $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$ *and a set of parameters* $\boldsymbol{\Theta} = \{\theta_i\}_{i=1}^n$*. It has the following properties:*

1. *No continuous variable is allowed be the parent of a discrete variable in* $\mathcal{G}$*.*
2. *Like in a BN, in the CLG model, to each discrete variable* X_i *in* $\mathbf{X_D}$ *is assigned its conditional probability table (CPT)* $\theta_i = P(X_i|\mathbf{Pa}_\mathcal{G}(X_i))$
3. *For any continuous variable* $\mathring{X}_i \in \mathring{\mathbf{X}}_\mathbf{C}$*,* θ_i *contain the parameters that represent the following conditional linear Gaussian distribution:*

$$P(\mathring{X}_i|\mathbf{Y} = \mathbf{y}, \mathring{\mathbf{Z}} = \mathring{\mathbf{z}}) = \mathcal{N}(\mathring{X}_i|\alpha(\mathbf{y}) + \boldsymbol{\beta}(\mathbf{y})^T\mathring{\mathbf{z}}, \sigma^2(\mathbf{y}))$$

where $\mathbf{Y}$ *and* $\mathring{\mathbf{Z}}$ *are the set of discrete and continuous parents of* $\mathring{X}_i$ *respectively.* $\alpha(\mathbf{y})$ *and* $\boldsymbol{\beta}(\mathbf{y})$ *are the coefficients of a linear regression model of* $\mathring{X}_i$ *given its continuous parents. These coefficients depend on the values* $\mathbf{y}$ *of the discrete parents.*

The product of all the CPTs and the conditional distributions represent the joint mixed distribution over $\mathbf{X}$*.*

In [Lauritzen, 1992, Lauritzen and Jensen, 2001] an algorithm is proposed in order to perform clique tree inference in CLG-BNs. The potentials associated to those cliques are functions expressed into their proposed canonical form, in which the marginalization and product operations are well defined. On the other hand, [Heckerman and Geiger, 1995] proposes a score for learning the BN structure of a CLG BN, which is called the *Be* (Bayesian equivalent) score. This score has the *BDe* score as a discrete component (using Dirichlet priors) and their proposed *BGe* (Bayesian likelihood equivalent) score as a continuous component (using normal-Wishart distribution as prior).

2.3 MTBF Models

As we have seen in the preceding section, Conditional Linear Gaussian models lack expressiveness, essentially because they represent a "large" multivariate Normal distribution. One simple way to extend this expressiveness is to not use a single Normal distribution but rather a mixture of distributions. This is the very key idea of MTBF-based models. In this section, we will review a few such popular models: first, we will describe *Mixture of Truncated Exponentials*, then we will present *Mixture of Truncated Polynomials* and, finally, the general model of *Mixture of Truncated Basis Functions*.

2.3.1 Mixture of Truncated Exponentials (MTE)

In the MTE model [Moral et al., 2001], the distribution over the set of all random variables $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$ is specified by a mixed probability distribution f such that:

- $\sum_{\mathbf{x_D} \in \Omega_{\mathbf{X_D}}} \int_{\Omega_{\mathring{\mathbf{X}}_\mathbf{C}}} f(\mathbf{x_D}, \mathring{\mathbf{x}}_\mathbf{C})\, d\mathring{\mathbf{x}}_\mathbf{C} = 1,$
- f is an MTE potential over $\mathbf{X}$, i.e.:

Definition 2.3.1 (MTE potential)
Let $\mathbf{Y} = \{X_{r_1}, \ldots, X_{r_p}\}$ and $\mathring{\mathbf{Z}} = \{\mathring{X}_{s_1}, \ldots, \mathring{X}_{s_q}\}$ be sets of discrete and continuous variables respectively. A function $\phi : \Omega_{\mathbf{Y} \cup \mathring{\mathbf{Z}}} \mapsto \mathbb{R}_0^+$ is a MTE potential if one of the two following conditions holds:

1. *ϕ can be written as:*

$$\phi(\mathbf{y}, \mathring{\mathbf{z}}) = a_0 + \sum_{i=1}^{m} a_i \exp\left\{ \sum_{j=1}^{p} b_i^{(j)} x_{r_j} + \sum_{k=1}^{q} b_i^{(p+k)} \mathring{x}_{s_k} \right\} \tag{2.20}$$

   for all $(x_{r_1}, \ldots, x_{r_p}) \in \mathbf{Y}$, $(\mathring{x}_{s_1}, \ldots, \mathring{x}_{s_q}) \in \mathring{\mathbf{Z}}$, where a_i, $i = 0, \ldots, m$ and $b_i^{(j)}$, $i = 1, \ldots, m$, $j = 1, \ldots, p + q$, are real numbers.

2. *There exists a partition $\Omega_1, \ldots, \Omega_k$ of $\Omega_{\mathbf{Y} \cup \mathring{\mathbf{Z}}}$ such that the domain of the continuous variables, $\Omega_{\mathring{\mathbf{Z}}}$, is divided into hypercubes, the domain $\Omega_\mathbf{Y}$ of the discrete variables is divided into arbitrary sets, and such that ϕ is defined as:*

$$\phi(\mathbf{y}, \mathring{\mathbf{z}}) = \phi_i(\mathbf{y}, \mathring{\mathbf{z}}) \quad \textit{if } (\mathbf{y}, \mathring{\mathbf{z}}) \in \Omega_i,$$

   where each ϕ_i, $i = 1, \ldots, k$, can be written in the form of Equation (2.20), i.e., it is a MTE potential on Ω_i.

MTEs present attractive features. First, they are expressive in the sense that they can approximate (w.r.t. the Kullback-Leibler distance) any continuous density function [Cobb et al., 2006, Cobb and Shenoy, 2006]. Second, they are easy to learn from datasets [Moral et al., 2002, Romero et al., 2004]. Finally, they satisfy Shafer-Shenoy's propagation axioms [Shenoy and Shafer, 1990] and inference can thus be performed using a junction tree-based algorithm [Moral et al., 2001, Cobb and Shenoy, 2006].

This algorithm can be described as follows. An undirected graph called a Markov network is first created: its nodes correspond to the variables of $\mathcal{X}$ and its edges are such that, for every MTE potential ϕ_i, all the nodes involved in ϕ_i are linked together. This graph is then triangulated by eliminating sequentially all the nodes. A node elimination consists

1. in adding edges to the Markov network in order to create a clique (a complete subgraph) containing the eliminated node and all its neighbors; and

2. in removing the eliminated node and its adjacent edges from the Markov network.

The cliques created during this process constitute the nodes of the junction tree. They are linked in order to satisfy a "running intersection" property [Madsen and Jensen, 1999]. Finally, each MTE potential ϕ_i is inserted into a clique containing all its variables.

A collect-distribute message-passing algorithm can then be performed in this junction tree, hence enabling to compute *a posteriori* marginal distributions of all the random variables. As usual, the message passed from one clique $\mathcal{C}_i$ to a neighbor $\mathcal{C}_j$ is the projection onto the variables in $\mathcal{C}_i \cap \mathcal{C}_j$ of the combination of the MTE potentials stored in $\mathcal{C}_i$ with the messages received by $\mathcal{C}_i$ from all its neighbors except $\mathcal{C}_j$. By Equation (2.20), combinations and projections are Algebraic operations over sums of exponentials. Unfortunately, these operations have a serious shortcoming: when propagating messages from one clique to another, the number of a_i/exp terms in Equation (2.20) tends to grow exponentially, hence limiting the use of this exact inference mechanism to problems with only a small number of cliques.

To overcome this issue, approximate algorithms based on MCMC [Moral et al., 2001] or on the Penniless algorithm [Rumían Salmerón, 2007] are provided in the literature.

2.3.2 Mixture of Truncated Polynomials

Mixtures of polynomials (MOP) are similar to MTE except that functions $\phi : \Omega_{\mathbf{Y} \cup \mathring{\mathbf{Z}}} \mapsto \mathbb{R}_0^+$ of Equation (2.20) are substituted by polynomials over the variables

in $\mathbf{Y} \cup \mathring{\mathbf{Z}}$ [Shenoy, 2011, Shenoy and West, 2011]. MOPs have several advantages over MTEs: their parameters for approximating density functions are easier to determine than those of MTEs. They are also applicable to a larger class of deterministic functions in hybrid BNs. As MTE, the MOP model satisfies Shafer-Shenoy's propagation axioms and inference can thus be performed by message-passing in a junction tree. But, similarly to Equation (2.20), the number of terms these messages involve tends to grow exponentially with the number of cliques in the junction tree, thereby limiting the use the message-passing algorithm to junction trees with a small number of cliques/random variables.

2.3.3 Mixture of Truncated Basis Functions

Mixtures of truncated basis functions (MTBF) generalize both MTEs and MOPs [Langseth et al., 2012b]. The definition of an MTBF is the same as Definition 2.3.1 except that Equation (2.20) is substituted by:

$$\phi(\mathbf{y}, \mathring{\mathbf{z}}) = \sum_{i=0}^{m} \prod_{k=1}^{q} a_{i,\mathbf{y}}^{(k)} \psi_i(\mathring{x}_{s_k}), \tag{2.21}$$

where potentials $\psi_i : \mathbb{R} \mapsto \mathbb{R}$ are basis functions. MTBFs are defined so that the potentials are closed under combination and projection which, again, ensures that inference can be performed by message-passing in a junction tree. By exploiting cleverly factorizations of terms in Equation (2.21), inference in MTBFs can be more efficient than in MTEs [Langseth et al., 2012a]. But, like all the other aforementioned models, the sizes of the messages tend to grow with the number of cliques in the junction tree.

2.4 Conclusion

In this chapter, we have recalled the usual models used to deal with both discrete and continuous random variables. First, we have presented models that discretize the continuous variables in order to fall back to only sets of discrete variables that can be dealt with using classical Bayesian networks. Then we focused on models that specify the density functions of the continuous variables. The first ones are Conditional Linear Gaussian (CLG) models, that exploit a multivariate Normal distribution. Then, we described mixture models, the most popular of which being MTEs, MOPs and MTBFs. It is worth noting that CLGs lack some expressiveness, i.e., there are many mixed probability distributions that cannot be very well approximated using CLGs (at least, without adding many latent variables). In addition, CLGs are provided with very efficient inference mechanisms. On the

other hand, MTEs, MOPs and MTBFs are very expressive but at the expense of inference tractability. Indeed, the number of algebraic operations they need to perform during inference tends to grow exponentially with the number of cliques in their junction trees.

Part II

Contributions

Chapter 3

Conditional Truncated Densities Bayesian Networks

The majority of Bayesian networks learning and inference algorithms rely on the assumption that all random variables are discrete, which is not necessarily the case in real-world problems. In situations where some variables are continuous, a trade-off between the expressive power of the model and the computational complexity of inference has to be done: on one hand, conditional Gaussian models are computationally efficient but they lack expressive power; on the other hand, mixtures of exponentials (MTE), bases or polynomials are expressive but this comes at the expense of tractability. In this chapter, we propose an alternative model that lies in between. It is composed of a "discrete" Bayesian network (BN) combined with a set of monodimensional conditional truncated densities modeling the uncertainty over the continuous random variables given their discrete counterpart resulting from a discretization process. We show that inference computation times in this new model are close to those in discrete BNs. Experiments confirm the tractability of the model and highlight its expressive power by comparing it with MTE.

3.1 Definition and properties

Discretizing continuous random variables raises two issues:

1. Which discretization function shall be used to minimize the loss of information? and
2. Will the loss of information affect significantly the results of inference?

A possible answer to the first question is to exploit "*conditional truncated densities*" [Cortijo and Gonzales, 2016]. The answer to the second question of course

strongly depends on the discretization performed but, as we shall see, conditional truncated densities can limit the discrepancy between the exact *a posteriori* marginal density functions of the continuous random variables and the approximation they provide.

Definition 3.1.1 (Conditional Truncated Density) *Let $\mathring{X}$ be a continuous random variable. Let $d_{\mathring{X}}$ be a discretization of $\mathring{X}$ with set of cutpoints $\{t_1, t_2, \ldots, t_g\}$. Finally, let X be a discrete random variable with domain $\Omega_X = \{0, \ldots, g\}$. A conditional truncated density is a function $f(\mathring{X}|X) : \Omega_{\mathring{X}} \times \Omega_X \mapsto \mathbb{R}_0^+$ satisfying the following properties:*

1. *$f(\mathring{x}|x) = 0$ for all $x \in \Omega_X$ and $\mathring{x} \notin [t_x, t_{x+1}]$ with, by abuse of notation $t_0 = \inf \Omega_{\mathring{X}}$ and $t_{g+1} = \sup \Omega_{\mathring{X}}$;*
2. *the following equation holds:*

$$\int_{t_x}^{t_{x+1}} f(\mathring{x}|x)\, d\mathring{x} = 1, \quad \textit{for all } x \in \Omega_X. \tag{3.1}$$

In other words, $f(\mathring{x}|x)$ represents the truncated density function of random variable $\mathring{X}$ over the interval of discretization $[t_x, t_{x+1}]$.

Lemma 3.1.1 *Let $P(X)$ be any probability distribution over the discrete random variable X of Definition 3.1.1. Then $f(\mathring{X}|X)P(X)$ is a mixed probability distribution over $\Omega_{\mathring{X}} \times \Omega_X$.*

Proof. By definition, $f(\mathring{X}|X)$ and $P(X)$ are non-negative real-valued functions, hence $f(\mathring{X}|X)P(X)$ is also a non-negative real-valued function. So, to prove that it is a mixed probability distribution, it is sufficient to show that:

$$\sum_{x \in \Omega_X} \int_{\Omega_{\mathring{X}}} f(\mathring{x}|x)P(x)\, d\mathring{x} = 1.$$

By Property 1., $f(\mathring{X} = \mathring{x}|X = x)P(X = x) = 0$ for all $x \in \Omega_X$ and $\mathring{x} \notin [t_x, t_{x+1}]$. So, the above equation is equivalent to:

$$\sum_{x \in \Omega_X} \int_{t_x}^{t_{x+1}} f(\mathring{x}|x)P(x)\, d\mathring{x} = 1,$$

which, by the fact that x is a constant inside the integral and by Eq. (3.1), is also equivalent to:

$$\sum_{x \in \Omega_X} P(x) \int_{t_x}^{t_{x+1}} f(\mathring{x}|x)\, d\mathring{x} = \sum_{x \in \Omega_X} P(x) = 1.$$

As a consequence, $f(\mathring{X}|X)P(X)$ is a mixed probability distribution. ■

Let us now introduce "*Conditionnal truncated densities Bayesian networks*" (ctdBN):

Definition 3.1.2 (Conditional truncated densities Bayesian networks) *Let* $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ *and* $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$ *be sets of discrete and continuous random variables respectively. Let* $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ *be a set of discretized variables resulting from the discretization of the variables in* $\mathring{\mathbf{X}}_\mathbf{C}$*. Then, a BN with conditional truncated densities is a pair* $(\mathcal{G}, \boldsymbol{\theta})$ *where:*

1. $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ *is a directed acyclic graph,*
2. $\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$ *and* $\mathbf{A}$ *is a set of arcs such that nodes* $\mathring{X}_i \in \mathring{\mathbf{X}}_\mathbf{C}$ have no children and exactly one parent equal to X_i *(this condition is the key to guarantee that inference in a ctdBN is as fast as in a classical BN),*
3. $\boldsymbol{\theta} = \boldsymbol{\theta}_\mathbf{D} \cup \boldsymbol{\theta}_\mathbf{C}$*,*
4. $\boldsymbol{\theta}_\mathbf{D} = \{P(X_i|\mathbf{Pa}(X_i))\}_{i=1}^n$ *is the set of the conditional probability tables of the discrete and discretized variables* X_i *in* $\mathcal{G}$ *given their parents* $\mathbf{Pa}(X_i)$ *in* $\mathcal{G}$*,*
5. $\boldsymbol{\theta}_\mathbf{C} = \{f(\mathring{X}_i|X_i)\}_{i=d+1}^n$ *is the set of the conditional truncated densities of the continuous random variables of* $\mathring{\mathbf{X}}_\mathbf{C}$*.*

Note that $\boldsymbol{\theta}_\mathbf{C}$ needs a very limited amount of memory compared to $\boldsymbol{\theta}_\mathbf{D}$ since truncated densities are monodimensional (e.g., a truncated normal distribution $f(\mathring{X}_i|X_i)$ is specified by only $2|\Omega_{X_i}|$ parameters).

An example of ctdBN is given in Figure 3.1. The model contains 3 continuous variables, $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_1, \mathring{X}_3, \mathring{X}_5\}$ represented by dotted circles, which are discretized into $\mathbf{X_C} = \{X_1, X_3, X_5\}$. Nodes in solid circles $\mathbf{X_C}$ and $\mathbf{X_D}$ form a classical BN. Finally, all the continuous nodes $\mathring{X}_i \in \mathring{\mathbf{X}}_\mathbf{C}$ are children of their discretized counterpart X_i and none has any child. The key idea of ctdBNs is thus to extend BNs by specifying the uncertainties over continuous random variables $\mathring{X}_i$ as 2-level functions: a "rough" probability distribution for discrete variable X_i and a finer-grain conditional density $f(\mathring{X}_i|X_i)$ for $\mathring{X}_i$. This idea can be somewhat related to second order probabilities [Baron, 1987].

Proposition 3.1.1 *In a BN with conditional truncated densities defined over* $\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$*, where* $\mathbf{X_D} = \{X_1, \ldots, X_d\}$*,* $\mathbf{X_C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$ *and* $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$*, function* $h : \mathbf{X} \mapsto \mathbb{R}_0^+$ *defined as:*

$$h(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \prod_{i=d+1}^{n} f(\mathring{X}_i|X_i) \tag{3.2}$$

is a mixed probability distribution over $\mathbf{X}$*.*

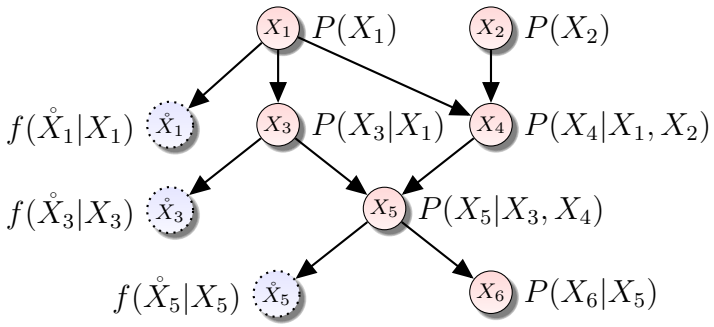


Figure 3.1: A BN with conditional truncated densities.

Proof. First, note that all the terms in the product are non-negative real-valued functions, hence h is also a non-negative real-valued function. Let

$$\begin{aligned}\alpha &= \sum_{x_1 \in X_1} \cdots \sum_{x_n \in X_n} \int_{\mathring{X}_{d+1}} \cdots \int_{\mathring{X}_n} \prod_{i=1}^{n} P(x_i|\mathbf{Pa}(x_i)) \prod_{i=d+1}^{n} f(\mathring{x}_i|x_i) d_{\mathring{x}_n} \cdots d_{\mathring{x}_{d+1}} \\ &= \sum_{x_1 \in X_1} \cdots \sum_{x_n \in X_n} \prod_{i=1}^{n} P(x_i|\mathbf{Pa}(x_i)) \times \\ &\quad \left(\int_{\mathring{X}_{d+1}} f(\mathring{x}_{d+1}|x_{d+1}) d_{\mathring{x}_{d+1}} \right) \cdots \left(\int_{\mathring{X}_n} f(\mathring{x}_n|x_n) d_{\mathring{x}_n} \right).\end{aligned}$$

By Property 2 of Definition 3.1.1, each integral of a conditional truncated density is equal to 1, hence:

$$\alpha = \sum_{x_1 \in X_1} \cdots \sum_{x_n \in X_n} \prod_{i=1}^{n} P(x_i|\mathbf{Pa}(x_i)).$$

This formula is also equal to 1 since its terms constitute a discrete BN. Therefore, $h(\mathbf{X})$ is a mixed probability distribution. ■

3.2 Representation properties

It was shown in [Cobb et al., 2006, Cobb and Shenoy, 2006] that MTEs can approximate standard probability density functions (w.r.t. the Kullback-Leibler distance). One may wonder whether ctdBNs are also faithful, i.e., whether they can provide good approximations of densities or mixed probability distributions. The propositions provided in this subsection show that the answer to this question is positive and that ctdBNs can actually approximate very general functions.

Proposition 3.2.1 *Let $\mathring{\mathbf{X}}_{\mathbf{C}} = \{\mathring{X}_1, ..., \mathring{X}_n\}$ be a set of continuous real-valued random variables of respective domains $\mathring{\Omega}_1, ..., \mathring{\Omega}_n$ such that none of the $\mathring{\Omega}_i$ is a singleton. Let $\mathring{\mathbf{\Omega}}_{\mathbf{C}} = \prod_{i=1}^{n} \mathring{\Omega}_i$ be the domain of $\mathring{\mathbf{X}}_{\mathbf{C}}$. Let $f : \mathring{\mathbf{\Omega}}_{\mathbf{C}} \mapsto \mathbb{R}$ be a probability density function. Assume that f is Lipschitz, i.e., there exists a constant $M > 0$ such that, for every pair $(\mathring{x}, \mathring{y})$ of elements of $\mathring{\mathbf{\Omega}}_{\mathbf{C}}$, $|f(\mathring{x}) - f(\mathring{y})| \leq M||\mathring{x} - \mathring{y}||$, where $||\mathring{x} - \mathring{y}||$ represents the L2-norm of vector $(\mathring{x} - \mathring{y})$.*

Then, for every strictly positive real number $\epsilon < 1$, there exists a ctdBN $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$ that approximates f up to ϵ, i.e.:

- *the nodes of Graph $\mathcal{G}$ are $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_{\mathbf{C}}$, where $\mathbf{X_D} = \{X_1, \ldots, X_n\}$ is a set of the discretized variables corresponding to $\mathring{\mathbf{X}}_{\mathbf{C}}$; in addition, let $\mathbf{\Omega_D} = \prod_{i=1}^{n} \Omega_i$ and $\mathbf{\Omega} = \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_{\mathbf{C}}$ be the domains of $\mathbf{X_D}$ and $\mathbf{X}$ respectively;*
- *$\mathcal{B}$ represents a mixed probability distribution $g : \mathbf{\Omega} \mapsto \mathbb{R}$ such that, for every $\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$, $|g(x, \mathring{x}) - f(\mathring{x})| \leq \epsilon$, where x corresponds to the discretized counterpart of $\mathring{x}$.*

Proof. [Proposition 3.2.1] Without loss of generality, we will assume in the sequel that $\epsilon \leq \min\{|\Omega_i| : i \in \{1, \ldots, n\}\}$, where $|\Omega_i|$ denotes the size of domain Ω_i. In addition, let us denote by $\mathbf{B}(\mathring{x}, r)$ the intersection of the hyperball of radius r centered on $\mathring{x}$ with $\mathring{\mathbf{\Omega}}_{\mathbf{C}}$. First, let us show that there exists $\mathring{a} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$ such that, for every $\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$, we have $f(\mathring{x}) \leq \epsilon$ whenever $||\mathring{x}|| \geq ||\mathring{a}||$. Proof by contradiction: assume that, for every $\mathring{a} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$, there exists $\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$ such that $||\mathring{x}|| \geq ||\mathring{a}||$ and $f(\mathring{x}) > \epsilon$. Then:

$$\int_{\mathring{\mathbf{\Omega}}_{\mathbf{C}}} f(\mathring{x}) d\mathring{x} = \int_{\mathbf{B}(0, ||\mathring{a}||)} f(\mathring{x}) d\mathring{x} + \int_{\{\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}} : ||\mathring{x}|| \geq ||\mathring{a}||\}} f(\mathring{x}) d\mathring{x}.$$

By hypothesis, there exists $\mathring{b} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}}$ such that $||\mathring{b}|| \geq ||\mathring{a}|| + \frac{\epsilon}{4M}$ and $f(\mathring{b}) > \epsilon$. So, since f is a probability density function, i.e., it is a positive function, the last term of the above equation is such that:

$$\int_{\{\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}} : ||\mathring{x}|| \geq ||\mathring{a}||\}} f(\mathring{x}) d\mathring{x} \geq \int_{\mathring{x} \in \mathbf{B}\left(\mathring{b}, \frac{\epsilon}{4M}\right)} f(\mathring{x}) d\mathring{x} + \int_{\{\mathring{x} \in \mathring{\mathbf{\Omega}}_{\mathbf{C}} : ||\mathring{x}|| \geq ||\mathring{b}|| + \frac{\epsilon}{4M}\}} f(\mathring{x}) d\mathring{x}$$

and since f is Lipschitz, for every $\mathring{x}$ inside Ball $\mathbf{B}\left(\mathring{b}, \frac{\epsilon}{4M}\right)$, we have that $|f(\mathring{x}) - f(\mathring{b})| \leq M||\mathring{x} - \mathring{y}|| \leq 2M\frac{\epsilon}{4M} = \frac{\epsilon}{2}$. As $f(\mathring{b}) > \epsilon$, we can deduce that $f(\mathring{x}) > \epsilon/2$ for every $\mathring{x} \in \mathbf{B}\left(\mathring{b}, \frac{\epsilon}{4M}\right)$ and, therefore, that the middle term in the above equation is greater than $\frac{\epsilon}{2} \int_{\mathring{x} \in \mathbf{B}\left(\mathring{b}, \frac{\epsilon}{4M}\right)} 1 d\mathring{x}$. This last integral corresponds to the volume of the intersection of the n-dimensional hyperball of radius $\frac{\epsilon}{4M}$ centered on $\mathring{b} = (\mathring{b}_1, \ldots, \mathring{b}_n)$ with $\mathring{\mathbf{\Omega}}_{\mathbf{C}}$. As $\epsilon \leq \min\{|\Omega_i|\}$, for each random variable $\mathring{X}_i$,

at least one interval among $(\mathring{b}_i - \frac{\epsilon}{4M}, \mathring{b}_i)$ and $(\mathring{b}_i, \mathring{b}_i + \frac{\epsilon}{4M})$ belongs to Ω_i. So the integral is greater than or equal to $1/2^n$ of the volume of an n-dimensional hyperball of radius r, which is equal to $\alpha = \pi^{n/2} r^n / \Gamma(\frac{n}{2} + 1)$. Consequently,

$$\int_{\mathring{\mathbf{\Omega}}_\mathbf{C}} f(\mathring{x}) d\mathring{x} > \int_{\mathbf{B}(0,||\mathring{a}||)} f(\mathring{x}) d\mathring{x} + \frac{\alpha\epsilon}{2^{n+1}} + \int_{\{\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C} : ||\mathring{x}|| \geq ||\mathring{b}|| + \frac{\epsilon}{4M}\}} f(\mathring{x}) d\mathring{x}.$$

By our contradiction hypothesis, the same process can be applied to the last term and, by induction, it is possible to construct an infinite sequence $\{\mathring{b}(i)\}_{i \geq 0}$ such that $\mathring{b}(0) = \mathring{b}$ and, for all $i \geq 1$, $||\mathring{b}(i)|| \geq ||\mathring{b}(i-1)|| + \frac{\epsilon}{4M}$ and $f(\mathring{b}(i)) > \epsilon$. Thus, for all $k > 0$,

$$\int_{\mathring{\mathbf{\Omega}}_\mathbf{C}} f(\mathring{x}) d\mathring{x} > \int_{\mathring{x} : ||\mathring{x}|| < ||\mathring{a}||} f(\mathring{x}) d\mathring{x} + k\frac{\alpha\epsilon}{2^{n+1}} + \int_{\{\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C} : ||\mathring{x}|| \geq ||\mathring{b}(k-1)|| + \frac{\epsilon}{4M}\}} f(\mathring{x}) d\mathring{x}.$$

So $\int_{\mathring{\mathbf{\Omega}}_\mathbf{C}} f(\mathring{x}) d\mathring{x}$ tends toward $+\infty$, which is impossible since f is a probability density function (hence its integral over $\mathring{\mathbf{\Omega}}_\mathbf{C}$ is equal to 1). Therefore, there necessarily exists $\mathring{a} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$ such that, for every $\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$, we have $f(\mathring{x}) \leq \epsilon$ whenever $||\mathring{x}|| \geq ||\mathring{a}||$.

Now, for any continuous variable $\mathring{X}_i$ of $\mathring{\mathbf{X}}_\mathbf{C}$, let $t_i^- = \max\{\inf \mathring{X}_i, -||\mathring{a}||\}$ and $t_i^+ = \min\{\sup \mathring{X}_i, ||\mathring{a}||\}$. Define a discretization function d_i of $\mathring{X}_i$ by its set of cutpoints $\{t_i^k\}$:

$$\left\{ t_i^k = t_i^- + k\frac{\epsilon}{\sqrt{n}M} : k \in \{0, \ldots, g_i\} \right\} \quad \text{with} \quad g_i = 1 + \left\lfloor \frac{\sqrt{n}M(t_i^+ - t_i^-)}{\epsilon} \right\rfloor.$$

Applying discretization function d_i to $\mathring{X}_i$, we obtain a discretized random variable X_i of domain Ω_i. Let $\mathbf{X_D}$ be the set of all these discretized variables and let $\mathbf{\Omega_D} = \prod_{i=1}^n \Omega_i$. Finally, for any value x_i of discretized variable X_i, denote by $\mathring{\mathbf{\Omega}}_{i|x_i}$ the subdomain of variable $\mathring{X}_i$ compatible with x_i, i.e., $\mathring{\mathbf{\Omega}}_{i|x_i} = [t_i^{x_i}, t_i^{x_i+1})$ if $x_i \notin \{0, g_i\}$, $\mathring{\mathbf{\Omega}}_{i|x_i} = \{\mathring{x}_i < t_i^0\}$ if $x_i = 0$ and $\mathring{\mathbf{\Omega}}_{i|x_i} = \{\mathring{x}_i \geq t_i^{g_i}\}$ if $x_i = g_i$. Let $\mathring{\mathbf{\Omega}}_{|x} = \prod_{i=1}^n \mathring{\mathbf{\Omega}}_{i|x_i}$.

We can now construct a joint probability distribution over $\mathbf{\Omega_D}$ and conditional truncated densities as follows: for every $x = (x_1, \ldots, x_n) \in \mathbf{\Omega_D}$, partition the set of indices $\{1, \ldots, n\}$ into $L = \{i : \mathring{\mathbf{\Omega}}_{i|x_i}$ is bounded$\}$, $L_{-\infty} = \{i : \inf \mathring{\mathbf{\Omega}}_{i|x_i} = -\infty\}$ and $L_{+\infty} = \{i : \sup \mathring{\mathbf{\Omega}}_{i|x_i} = +\infty\}$. Fix the joint probability value of $x = (x_1, \ldots, x_n)$ to $P(x) = \int_{\mathring{\mathbf{\Omega}}_{|x}} f(\mathring{x}) d\mathring{x}$ and define for all $\mathring{x}_i \in \mathring{\mathbf{\Omega}}_{|x_i}$ the truncated conditional density function $h(\mathring{x}_i|x_i)$ as:

$$h(\mathring{x}_i|x_i) = \begin{cases} \left(\frac{\epsilon}{\beta}\right)^n e^{-\left\{\frac{\pi}{4}\left(\frac{\epsilon}{\beta}\right)^{2n} (\mathring{x}_i - t_i^0)^2\right\}} & \text{if } i \in L_{-\infty} \\ \sqrt{n}M/\epsilon & \text{if } i \in L \\ \left(\frac{\epsilon}{\beta}\right)^n e^{-\left\{\frac{\pi}{4}\left(\frac{\epsilon}{\beta}\right)^{2n} (\mathring{x}_i - t_i^{g_i})^2\right\}} & \text{if } i \in L_{+\infty} \end{cases}$$

where $\beta = \max\{1, \sqrt{n}M\}$. Then, it is easy to see that $P(\cdot)$ is non-negative and that $\sum_{x \in \mathbf{\Omega_D}} P(x) = \int_{\mathring{\mathbf{\Omega}}_\mathbf{C}} f(\mathring{x}) d\mathring{x} = 1$. In addition, $\int_{\mathring{\mathbf{\Omega}}_{|x_i}} h(\mathring{x}_i|x_i) d\mathring{x}_i = 1$ for all x_i since the formulas for $i \in L_{-\infty} \cup L_{+\infty}$ are nothing else than twice the density function of a Normal distribution of variance $\sqrt{\frac{2}{\pi}} \left(\frac{\beta}{\epsilon}\right)^n$. So for all x_i, since $h(\mathring{x}_i|x_i) \geq 0$, h is a truncated conditional density function. Consequently $P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i)$ defines a mixed probability distribution.

Now, let us show that, for all $\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$, $|f(\mathring{x}) - P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i)| \leq \epsilon$, where x is the discretized value of $\mathring{x}$. Consider any element $\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$. First, assume that $L_{+\infty} \cup L_{-\infty} \neq \emptyset$, then $||\mathring{x}|| \geq ||\mathring{a}||$ and, consequently, $f(\mathring{x}) \leq \epsilon$. By definition, $P(x) \leq 1$. In addition, for all $i \in L_{+\infty} \cup L_{-\infty}$, $h(\mathring{x}_i|x_i) \leq \left(\frac{\epsilon}{\beta}\right)^n$. So, if $N_\infty = |L_{-\infty}| + |L_{+\infty}|$, then:

$$P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i)| \leq \left(\frac{\epsilon}{\beta}\right)^{nN_\infty} \times \left(\frac{\sqrt{n}M}{\epsilon}\right)^{n-N_\infty}.$$

As $N_\infty \geq 1$, $nN_\infty \geq n \geq n - N_\infty + 1$. Hence, as $\beta = \max\{1, \sqrt{n}M\}$ and $\epsilon/\beta \leq \epsilon < 1$,

$$P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i)| \leq \epsilon \times \left(\frac{\epsilon}{\beta}\right)^{n-N_\infty} \times \left(\frac{\sqrt{n}M}{\epsilon}\right)^{n-N_\infty} \leq \epsilon.$$

If, on the contrary, $L_{+\infty} \cup L_{-\infty} = \emptyset$, then all the $\Omega_{i|x_i}$ are bounded and their sizes are all equal to $\epsilon/(\sqrt{n}M)$, so $\mathring{\mathbf{\Omega}}_{|x}$ is also bounded. Let $f^- = \min_{\mathring{x} \in \mathring{\mathbf{\Omega}}_{|x}} f(\mathring{x})$ and $f^+ = \max_{\mathring{x} \in \mathring{\mathbf{\Omega}}_{|x}} f(\mathring{x})$. Then:

$$\int_{\Omega_{i|x_i}} f^- dx = \left(\frac{\epsilon}{\sqrt{n}M}\right)^n f^- \leq P(x) \leq \left(\frac{\epsilon}{\sqrt{n}M}\right)^n f^+ = \int_{\Omega_{i|x_i}} f^+ dx.$$

As all the $h(\mathring{x}_i|x_i)$'s are equal to $\sqrt{n}M/\epsilon$, we have $f^- \leq P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i) \leq f^+$. Now, for any pair of elements $(\mathring{y}, \mathring{z})$ of $\mathring{\mathbf{\Omega}}_{|x}$, $||\mathring{y} - \mathring{z}|| < \sqrt{n(\epsilon/\sqrt{n}M)^2} = \frac{\epsilon}{M}$. So, as f is Lipschitz, $|f(\mathring{y}) - f(\mathring{z})| \leq M\frac{\epsilon}{M} = \epsilon$. As a consequence, $f^+ - f(\mathring{x}) \leq \epsilon$ and $f(\mathring{x}) - f^- \leq \epsilon$. Hence, $|f(\mathring{x}) - P(x) \prod_{i=1}^n h(\mathring{x}_i|x_i)| \leq \epsilon$.

To complete the proof, note that any joint distribution $P(X_1, \ldots, X_n)$ can be rewritten as $P(X_1, \ldots, X_n) = P(X_1) \prod_{i=2}^n P(X_i|X_1, \ldots, X_{i-1})$. Using this decomposition, we obtain a ctdBN whose discrete and continuous nodes are $\{X_1, \ldots, X_n\}$ and $\{\mathring{X}_1, \ldots, \mathring{X}_n\}$ respectively, and in which the parents of discretized node X_i are the set $\{X_1, \ldots, X_{i-1}\}$, the conditional probability $P(X_i|X_1, \ldots, X_{i-1})$ resulting from the joint probability $P(X_1, \ldots, X_n)$ are defined in the above paragraphs. Finally, to each X_i is assigned as a child a continuous node $\mathring{X}_i$ whose conditional truncated density is $h(\mathring{X}_i|X_i)$ as defined in the above paragraph. As shown above, this ctdBN approximates f up to ϵ. ■

The above proposition shows that any Lipschitz multivariate probability density function can be (arbitrarily well) approximated by a ctdBN. Note that, to do so, the conditional truncated densities used by such ctdBN need not be "complex": in the proof of this proposition, only uniform and conditional truncated normal distributions were used. An obvious corollary of this proposition is that standard density functions can be approximated by ctdBNs:

Corollary 3.2.1 *Standard distributions like, e.g., univariate and multivariate Normal distributions, Beta distributions $B(\mathring{x}, \alpha, \beta)$, with $\alpha, \beta \geq 2$, Gamma distribution $\Gamma(\mathring{x}, \alpha, \beta)$ with $\alpha > 2$, as well as their combinations by mutually independent random variables, can be approximated up to $\epsilon < 1$ by ctdBNs.*

Proof. [Corollary 3.2.1] The absolute value of the derivative of the density function of a univariate normal distribution is highest at its inflection point, which corresponds to $x = \mu \pm \sigma$. At that point, the derivative is equal to $M = \exp(-0.5)/(\sqrt{2\pi}\sigma^2)$. So the univariate normal distribution is Lipschitz. By Proposition 3.2.1, it can be approximated up to ϵ by a ctdBN.

The density function of a multivariate normal distribution is equal to:

$$f(\mathring{x}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} \exp\left[-\frac{1}{2}(\mathring{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathring{x} - \boldsymbol{\mu})\right].$$

$\boldsymbol{\Sigma}^{-1}$ being invertible, it is diagonalizable and its eigenvalues are all different from 0. By expressing $\mathring{x}$ in the basis of the eigenvectors, $f(\mathring{x})$ becomes a product of univariate normal distributions and the preceding paragraph implies that f is Lipschitz and can be approximated up to ϵ by a ctdBN.

The density of the Beta distribution is $f(\mathring{x}) = \mathring{x}^{\alpha-1}(1-\mathring{x})^{\beta-1}/B(\alpha,\beta)$, with $B(\alpha,\beta) = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$. The derivative is therefore equal to $f'(\mathring{x}) = [(\alpha-1)(1-x) - (\beta-1)x](\alpha-1)x^{\alpha-2}(1-x)^{\beta-2}/B(\alpha,\beta)$ If $2 = \alpha < \beta$, then $|f'(\mathring{x})|$ is maximal when $\mathring{x} = 0$ and it is equal to $(\alpha-1)/B(\alpha,\beta)$. If $2 = \beta < \alpha$, then $|f'(\mathring{x})|$ is maximal when $\mathring{x} = 1$ and it is equal to $(\beta-1)/B(\alpha,\beta)$. Finally, if $2 < \alpha$ and $2 < \beta$, it is known that the Beta distribution is bell-shaped with two inflection points at $\mathring{x} = (\alpha - 1 \pm \sqrt{\frac{(\alpha-1)(\beta-1)}{\alpha+\beta-3}})/(\alpha+\beta-1)$. So the derivative is bounded and f is Lipschitz.

When $\alpha > 2$, the Gamma distribution is bell-shaped and its inflection points are $\beta(\alpha - 1 \pm \sqrt{\alpha-1})$. Hence, the distribution is Lipschitz.

To complete the proof, consider a set $\mathring{\mathbf{X}}_{\mathbf{C}}$ of sets of random variables $\mathring{\mathbf{X}}_{\mathbf{C}} = \{\mathring{\mathbf{X}}_1, \ldots, \mathring{\mathbf{X}}_n\}$ such that all the $\mathring{\mathbf{X}}_i$'s are mutually independent. Let $f_i, i = 1, \ldots, n$, be the respective density functions of the $\mathring{\mathbf{X}}_i$'s and assume that all the f_i's belong to the probability density functions defined in the above paragraphs. Then, every

f_i can be approximated up to $\epsilon_0 = \epsilon/2^n$ by some ctdBN $\mathcal{B}_i$ defining a mixed probability distribution g_i, i.e., for any $\mathring{x}_i \in \mathring{\mathbf{X}}_i$, $|f_i(\mathring{x}_i) - g_i(x_i, \mathring{x}_i)| \leq \epsilon_0$, where x_i correspond to the discretized value of $\mathring{x}_i$. Now, the joint density function $f : \mathring{\mathbf{X}}_\mathbf{C} \mapsto \mathbb{R}$ is defined as $f(\mathring{x}) = \prod_{i=1}^{n} f_i(x_i)$, for all $x = (x_1, \ldots, x_n)$ since the $\mathring{\mathbf{X}}_i$'s are mutually independent. Let $\mathcal{B}$ be the ctdBN resulting from the union of all the $\mathcal{B}_i$'s, i.e., its graphical structure is the union of all the graphical structures of the $\mathcal{B}_i$'s and $\mathcal{B}$ represents mixed probability $g(x, \mathring{x}) = \prod_{i=1}^{n} g_i(x_i, \mathring{x}_i)$. So, we have that $g(x, \mathring{x}) \leq \prod_{i=1}^{n}(f_i(\mathring{x}_i) + \epsilon_0)$. Let $\mathcal{S}_k$ be the set of k-subsets of $\{1, \ldots, n\}$. Then:

$$\prod_{i=1}^{n}(f_i(\mathring{x}_i) + \epsilon_0) = \prod_{i=1}^{n} f_i(\mathring{x}_i) + \sum_{k=0}^{n-1} \sum_{S \in \mathcal{S}_k} \left(\epsilon_0^{n-k} \prod_{j \in S} f_j(\mathring{x}_j) \right). \tag{3.3}$$

As the f_j's are probability density functions, $\prod_{j \in S} f_j(\mathring{x}_j) \leq 1$. In addition, for every n, k, we have that $\epsilon_0^{n-k} \leq \epsilon_0$ since $\epsilon_0 < 1$. Finally, $\sum_{k=0}^{n-1} \sum_{S \in \mathcal{S}_k} 1 < 2^n$ since this corresponds to the size minus 1 of the power set of $\{1, \ldots, n\}$. So, the right hand side of Equation (3.3) is less than or equal to $\prod_{i=1}^{n} f_i(\mathring{x}_i) + 2^n \epsilon_0 = f(\mathring{x}) + \epsilon$. We can show similarly, that $g(x, \mathring{x}) \geq f(\mathring{x}) - \epsilon$. So ctdBN $\mathcal{B}$ approximates up to ϵ probability density function f. ■

But ctdBNs represent compactly the uncertainties over both discrete and continuous random variables. So, they may also provide good approximations of mixed probability distributions and the following proposition justifies this intuition:

Proposition 3.2.2 *Let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ be a set of discrete random variables of respective domains $\{\Omega_1, \ldots, \Omega_d\}$ and let $\mathbf{\Omega_D} = \prod_{i=1}^{d} \Omega_i$ be the domain of $\mathbf{X_D}$. Let $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, ..., \mathring{X}_n\}$ be a set of continuous random variables of respective domains $\mathring{\Omega}_{d+1}, ..., \mathring{\Omega}_n$ such that none of the $\mathring{\Omega}_i$ is a singleton. Let $\mathring{\mathbf{\Omega}}_\mathbf{C} = \prod_{i=d+1}^{n} \mathring{\Omega}_i$ be the domain of $\mathring{\mathbf{X}}_\mathbf{C}$. Finally, let $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$ and $\mathbf{\Omega} = \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$.*

Let $f : \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C} \mapsto \mathbb{R}$ be a mixed probability distribution. Assume that f is Lipschitz w.r.t. the continuous variables of $\mathring{\mathbf{X}}_\mathbf{C}$, i.e., there exists a constant $M > 0$ such that, for every pair $(\mathring{x}, \mathring{y})$ of elements of $\mathring{\mathbf{\Omega}}$ such that $x_i = y_i$ for all $i \in \{1, \ldots, d\}$, $|f(\mathring{x}) - f(\mathring{y})| \leq M||\mathring{x} - \mathring{y}||$, where $||\mathring{x} - \mathring{y}||$ represents the L2-norm of vector $(\mathring{x} - \mathring{y})$.

Then, for every strictly positive real number $\epsilon < 1$, there exists a ctdBN $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$ that approximates f up to ϵ, i.e.:

- *the nodes of $\mathcal{G}$ are $\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$, where $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ is a set of discretized variables corresponding to $\mathring{\mathbf{X}}_\mathbf{C}$; in addition, let $\mathbf{\Omega_C} = \prod_{i=d+1}^{n} \Omega_i$ and $\mathbf{\Omega} = \mathbf{\Omega_D} \times \mathbf{\Omega_C} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$ be the domains of $\mathbf{X_C}$ and $\mathbf{X}$ respectively;*

- *$\mathcal{B}$ represents a mixed probability density function $g : \mathbf{\Omega} \mapsto \mathbb{R}$ such that, for every $(y, \mathring{x}) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$, $|g(y, x, \mathring{x}) - f(y, \mathring{x})| \leq \epsilon$, where x corresponds to the discretized counterpart of $\mathring{x}$.*

Proof. [Proposition 3.2.2] If $\mathbf{X_D} = \emptyset$, then Proposition 3.2.2 exactly corresponds to Proposition 3.2.1. So, assume that $\mathbf{X_D} \neq \emptyset$.

For every $y = (y_1, \ldots, y_d) \in \mathbf{\Omega_D}$, let $\pi(y) = \int_{\mathring{x} \in \mathring{\mathbf{X}}_\mathbf{C}} f(y, \mathring{x}) d\mathring{x}$. As y corresponds to the discrete part of f, $\pi(y)$ corresponds to the probability of y w.r.t. f. So $k_y : \mathring{\mathbf{\Omega}}_\mathbf{C} \mapsto \mathbb{R}$ defined as $k_y(\mathring{x}) = f(y, \mathring{x})/\pi(y)$ for all $\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$ is a probability density function. In addition, by the hypotheses of Proposition 3.2.2, it is Lipschitz. Hence the proof of Proposition 3.2.1 can be applied on it. Let us call $\mathring{a}_y$ the vector $\mathring{a}$ of this proof applied on $k_y(\cdot)$. In addition, let $\mathring{a}$ denote a vector of $\{\mathring{a}_y : y \in \mathbf{\Omega_D}\}$ with the highest L2-norm. Then, for every $y \in \mathbf{\Omega_D}$ and any $\mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C}$, we have $k_y(\mathring{x}) \leq \epsilon$ whenever $||\mathring{x}|| \geq ||\mathring{a}||$. Applying the proof of Proposition 3.2.1, with this value of $\mathring{a}$, we can therefore perform the same discretization of $\mathring{x}$ into x and construct the same conditional truncated density functions $h(\mathring{x}|x)$ for all the values of y. The proof of Proposition 3.2.1 also shows that, by setting $P_y(x) = \int_{\mathring{\mathbf{\Omega}}_{|x}} k_y(\mathring{x}) d\mathring{x}$, then we have that:

$$\left| k_y(\mathring{x}) - P_y(x) \prod_{i=d+1}^{n} h(\mathring{x}_i|x_i) \right| \leq \epsilon, \quad \text{for all } \mathring{x} \in \mathring{\mathbf{\Omega}}_\mathbf{C} \text{ and all } y \in \mathbf{\Omega_D}, \qquad (3.4)$$

and $\sum_x P_y(x) = 1$ for every y. In other words, $P_y(x)$ corresponds to a conditional distribution of x given y. Define $P(y, x) = P_y(x) \times \pi(y)$. Then, $P(y, x)$ corresponds to the joint distribution of x and y (i.e., $\sum_{x,y} P(y, x) = 1$). So, as $k_y(\mathring{x}) = f(y, \mathring{x})/\pi(y)$ and $\pi(y) \leq 1$ (since it is a probability), Equation (3.4) implies that:

$$\left| f(y, \mathring{x}) - P(y, x) \prod_{i=d+1}^{n} h(\mathring{x}_i|x_i) \right| \leq \epsilon\pi(y) \leq \epsilon, \quad \text{for all } (y, \mathring{x}) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}.$$

The completion of the proof is now the same as that of Proposition 3.2.1: joint distribution $P(y, x)$ can be decomposed as $P(y_1) \times \prod_{i=2}^{d} P(y_i|y_1, \ldots, y_{i-1}) \times P(x_{d+1}|y_1, \ldots, y_d) \prod_{j=d+2}^{n} \times P(x_j|y_1, \ldots, y_d, x_{d+1}, \ldots, x_{j-1})$ and the resulting ctdBN follows. ■

CtdBNs also have some decomposability properties. For instance, the following proposition shows that, if the mixed probability distribution to be approximated is decomposable, then so is also the approximating ctdBN:

Proposition 3.2.3 *Let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ and $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, ..., \mathring{X}_n\}$ be sets of discrete and continuous random variables respectively. Let $f : \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C} \mapsto \mathbb{R}$ be a mixed probability distribution. Assume that sets of variables $\mathbf{X_D}$ and $\mathring{\mathbf{X}}_\mathbf{C}$ can be partitioned into sets $\{\mathbf{X}_{D_1}, \ldots, \mathbf{X}_{D_k}\}$ and $\mathring{\mathbf{X}}_{C_1}, \ldots, \mathring{\mathbf{X}}_{C_k}$ respectively, and that there exist some non-negative functions $f_i : \mathbf{\Omega}_{D_i} \times \mathring{\mathbf{\Omega}}_{C_i} \mapsto \mathbb{R}$, $i = 1, \ldots, k$, such that $f(x, \mathring{x}) = \prod_{i=1}^{k} f_i(x_{D_i}, \mathring{x}_{C_i})$ for all $(x, \mathring{x}) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$. Then if f is Lipschitz w.r.t. the continuous variables of $\mathring{\mathbf{X}}_\mathbf{C}$, it can be approximated up to ϵ by a ctdBN which has the same decomposition, i.e., sets $(\mathbf{X}_{D_i} \cup \mathbf{X}_{C_i} \cup \mathring{\mathbf{X}}_{C_i})$, $i = 1, \ldots, k$, where $\mathbf{X}_{C_i}$ are the discretized counterparts of $\mathring{\mathbf{X}}_{C_i}$, form the connected components of the ctdBN's graphical structure.*

Proof. [Proposition 3.2.3] According to the proof of Proposition 3.2.2, since f is Lipschitz, the continuous variables $\mathring{X}_i$ of $\mathring{\mathbf{X}}_\mathbf{C}$ can be discretized into discrete variables X_i of $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ and, for all $(y, \mathring{x}) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$, $f(y, \mathring{x})$ can be approximated up to ϵ by $P(y, x) \prod_{i=d+1}^{n} h(\mathring{x}_i | x_i)$, where x is the discretized counterpart of $\mathring{x}$ and $P(y, x)$ is the joint probability distribution defined as:

$$P(y, x) = \int_{\mathring{\mathbf{\Omega}}_{|x}} f(y, \mathring{x}) d\mathring{x}. \tag{3.5}$$

Let us show that this joint distribution can be decomposed w.r.t. sets $\mathcal{C}_i = \mathbf{X}_{D_i} \cup \mathbf{X}_{C_i} \cup \mathring{\mathbf{X}}_{C_i}$, $i = 1, \ldots, k$. Proof by induction on i: let $i = 1$ and let $\mathbf{X}_{E_1} = \mathbf{X_D} \setminus \mathbf{X}_{C_1}$ and $\mathring{\mathbf{X}}_{E_1} = \mathring{\mathbf{X}}_\mathbf{C} \setminus \mathring{\mathbf{X}}_{C_1}$. Function f can be decomposed as $f(y, \mathring{x}) = f_1(y_{C_1}, \mathring{x}_{C_1}) \times h_1(y_{E_1}, \mathring{x}_{E_1})$, with $h_1(y_{E_1}, \mathring{x}_{E_1}) = \prod_{j=2}^{k} f_j(y_{C_j}, \mathring{x}_{C_j})$. Note that f_1 and h_1 share no variable in common. Then Equation (3.5) is equal to:

$$\begin{aligned} P(y, x) &= \int_{\mathring{\mathbf{\Omega}}_{|x_{C_1}}} \int_{\mathring{\mathbf{\Omega}}_{|x_{E_1}}} f_1(y_{C_1}, \mathring{x}_{C_1}) \times h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1} d\mathring{x}_{C_1} \\ &= \int_{\mathring{\mathbf{\Omega}}_{|x_{C_1}}} f_1(y_{C_1}, \mathring{x}_{C_1}) d\mathring{x}_{C_1} \int_{\mathring{\mathbf{\Omega}}_{|x_{E_1}}} h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1} \end{aligned}$$

Now, we also have that:

$$\begin{aligned} P(y_{C_1}, x_{C_1}) &= \sum_{y_{E_1}} \sum_{x_{E_1}} P(y, x) \\ &= \int_{\mathring{\mathbf{\Omega}}_{|x_{C_1}}} f_1(y_{C_1}, \mathring{x}_{C_1}) d\mathring{x}_{C_1} \sum_{y_{E_1}} \sum_{x_{E_1}} \int_{\mathring{\mathbf{\Omega}}_{|x_{E_1}}} h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1} \\ &= \int_{\mathring{\mathbf{\Omega}}_{|x_{C_1}}} f_1(y_{C_1}, \mathring{x}_{C_1}) d\mathring{x}_{C_1} \sum_{y_{E_1}} \int_{\mathring{\mathbf{\Omega}}_{E_1}} h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1} \end{aligned}$$

Note that, by definition, $h_1(y_{E_1}, \mathring{x}_{E_1}) = \prod_{j=2}^{k} f_j(y_{C_j}, \mathring{x}_{C_j})$ is a mixed probability

distribution over $\mathbf{\Omega}_{E_1} \times \mathring{\mathbf{\Omega}}_{E_1}$. Consequently, we have that:

$$\sum_{y_{E_1}} \int_{\mathring{\mathbf{\Omega}}_{E_1}} h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1} = 1,$$

and, therefore, that $P(y_{C_1}, x_{C_1}) = \int_{\mathring{\mathbf{\Omega}}_{|x_{C_1}}} f_1(y_{C_1}, \mathring{x}_{C_1}) d\mathring{x}_{C_1}$. We can prove in a similar way that $P(y_{E_1}, x_{E_1}) = \int_{\mathring{\mathbf{\Omega}}_{|x_{E_1}}} h_1(y_{E_1}, \mathring{x}_{E_1}) d\mathring{x}_{E_1}$. Consequently, $P(y, x) = P(y_{C_1}, x_{C_1}) P(y_{E_1}, x_{E_1})$. So, P can be decomposed similarly to f as concerns clique $\mathcal{C}_1$. By induction, we can repeat the same process with mixed probability h_1 rather than f and the result follows. ■

3.3 Inference in CtdBNs

The terms in Equation (3.2) satisfy Shafer-Shenoy's propagation axioms [Shenoy and Shafer, 1990], so we can rely on a message-passing algorithm in a junction tree to perform inference. The latter is constructed by node eliminations from the moral/induced graph (see Section 1.4.2). It was proved that first eliminating all simplicial nodes, i.e., nodes that, together with their neighbors in the moral/induced graph, constitute a clique (a complete maximal subgraph), cannot prevent obtaining a junction tree that is optimal w.r.t. inference [van den Eijkhof and Bodlaender, 2002]. By the definition of ctdBNs, all the continuous nodes $\mathring{X}_i \in \mathring{\mathbf{X}}_{\mathbf{C}}$ constitute a clique with their parent X_i (for instance, in Figure 3.1, $\{X_3, \mathring{X}_3\}$ is a complete maximal subgraph and is thus a clique). As a consequence, the junction tree of a ctdBN is simply the junction tree of its discrete BN part defined over $\mathbf{X}_{\mathbf{C}} \cup \mathbf{X}_{\mathbf{D}}$ to which cliques $\{X_i, \mathring{X}_i\}$, for $\mathring{X}_i \in \mathring{\mathbf{X}}_{\mathbf{C}}$, have been added (linked to a clique containing X_i in order to satisfy the running intersection property). Figure 3.2 shows a junction tree related to the ctdBN of Figure 3.1. All the CPTs $P(X_i|\mathbf{Pa}(X_i))$, $i = 1, \ldots, n$, are inserted into cliques not containing any continuous node of $\mathring{\mathbf{X}}_{\mathbf{C}}$. Of course, conditional truncated densities are inserted into cliques $\{X_i, \mathring{X}_i\}$, $\mathring{X}_i \in \mathring{\mathbf{X}}_{\mathbf{C}}$.

The inference process can now be performed by message passing within the junction tree, for instance using a usual collect-distribute algorithm in a Shafer-Shenoy-like architecture [Shafer, 1996], sending messages in both directions on all the edges of the junction tree.

There remains to show how to compute the messages sent on the separators in the Shafer-Shenoy collect-distribute algorithm and how to encode and insert evidence into junction tree $\mathcal{T}$. First, let us address the second problem. Of course evidence on discrete random variables X_i are handled in a usual manner by multiplying the joint mixed probability distribution $g(X, \mathring{X})$ represented by the ctdBN

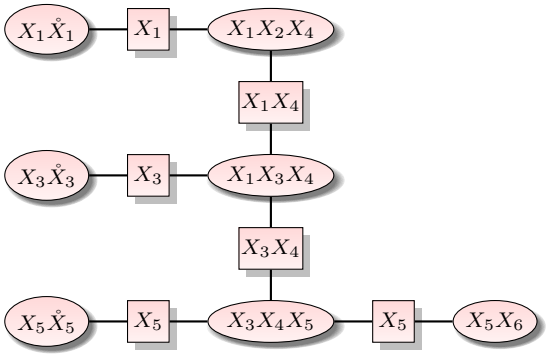


Figure 3.2: A junction tree for the ctdBN of Figure 3.1.

with discrete beliefs of the type $P(e_{X_i}|X_i)$. This corresponds to adding probability table $P(e_{X_i}|X_i)$ into a clique of $\mathcal{T}$ containing X_i. For continuous random variables, two cases can occur: first, it may be the case that the available evidence on continuous random variable $\mathring{X}_i$ can be encoded as an evidence on its discretized random variable, then we do so. For instance, if $\{t_1, \ldots, t_{g_i}\}$ are the cutpoints of the discretization function applied to $\mathring{X}_i$, then evidence "$\mathring{X}_i$ is known to belong $[t_j, t_k]$" can be encoded as a vector $P(e_{X_i}|X_i)$ whose values are 1 for the indices in $\{j, \ldots, k-1\}$, else 0. Second, it may be impossible to enter evidence $e_{\mathring{X}_i}$ on $\mathring{X}_i$ into X_i. In this case, $e_{\mathring{X}_i}$ can be of the type "$\mathring{X}_i$ belongs to some interval $[a, b]$", with $a, b \notin \{t_1, \ldots, t_{g_i}\}$. Such an evidence can be represented by function $f_i(e_{\mathring{X}_i}|\mathring{X}_i) : \Omega_{\mathring{X}_i} \mapsto [0, 1]$ equal to 1 when $\mathring{X}_i \in [a, b]$ and 0 otherwise. As function f_i is defined only over $\mathring{X}_i$, it can be entered into the clique $\mathcal{C}_i = \{X_i, \mathring{X}_i\}$ of junction tree $\mathcal{T}$. More generally, beliefs about $\mathring{X}_i$ can be entered as any $[0, 1]$-valued function $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ into clique $\mathcal{C}_i$. It is easy to see that the product of the evidence functions $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ and $P(e_{X_i}|X_i)$ with $g(X, \mathring{X})$ defines, up to a proportional constant equal to the probability of all the evidence, a new mixed probability distribution.

Now, there remains to show how to compute the messages sent from one clique, say $\mathcal{C}_i$, to one of its neighbor $\mathcal{C}_j$. Two cases can occur. First, assume that $\mathcal{C}_i$ contains a continuous random variable $\mathring{X}_i$. Then, by construction of ctdBNs, $\mathcal{C}_i = \{X_i, \mathring{X}_i\}$, with X_i the discretized variable corresponding to $\mathring{X}_i$. By construction, clique $\mathcal{C}_i$ has only one neighbor clique, say $\mathcal{C}_j$, and the separator between $\mathcal{C}_i$ and $\mathcal{C}_j$ is necessarily $S_{ij} = \{X_i\}$. Clique $\mathcal{C}_i$ contains only conditional truncated density $g_i(\mathring{X}_i|X_i)$ and, potentially, some evidence belief $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$. So, in order to remove variable $\mathring{X}_i$ from the equations, it must be marginalized out as:

$$\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}(x_i) = \int_{\Omega_{\mathring{X}_i}} g_i(\mathring{x}_i|x_i) f_i(e_{\mathring{X}_i}|\mathring{x}_i)\, d\mathring{x}_i, \text{ for all } x_i \in \Omega_{X_i}. \tag{3.6}$$

Assume that $\{t_1, \ldots, t_{g_i}\}$ are the cutpoints of the discretization function applied to $\mathring{X}_i$. Then message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ is a real-valued vector of size $g_i + 1$. So, messages sent from cliques containing continuous random variables to their neighbor are necessarily vectors of finite size. In addition, whether $\mathring{X}_i$ received evidence or not, note that message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ is computed by integrating a *univariate* function, which, in practice, is not time consuming (it can be done either exactly in closed-form formula or approximately using a MCMC algorithm or well-known tables like for normal distributions).

The second case for computing messages concerns situations in which clique $\mathcal{C}_i$ contains only discrete random variables. Then, by construction, the separator $\mathcal{S}_{ij} = \mathcal{C}_i \cap \mathcal{C}_j$ contains only discrete random variables. Message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ can therefore be computed as usual by first multiplying all the messages sent to $\mathcal{C}_i$ by all $\mathcal{C}_i$'s neighbors except $\mathcal{C}_j$ with the product of the CPTs stored into $\mathcal{C}_i$, and then by marginalizing out all the variables in $\mathcal{C}_i \setminus \mathcal{C}_j$.

Proposition 3.3.1 *Let e represent all the evidence entered into junction tree $\mathcal{T}$. Assume that Shafer-Shenoy's message-passing algorithm has been performed, with messages computed as described above.*

Let $\mathcal{C}_k$ be any clique containing only discrete variables and let $\mathcal{C}_{i_1}, \ldots, \mathcal{C}_{i_r}$ be the neighbors of $\mathcal{C}_k$. Then the CPT resulting from the normalization of the product of all the messages $\mathcal{M}_{\mathcal{C}_{i_j} \to \mathcal{C}_k}$, $j = 1, \ldots, r$, with the CPTs stored into $\mathcal{C}_k$ is equal to the joint posterior distribution of the variables of $\mathcal{C}_k$ given evidence e. The posterior of any variable in $\mathcal{C}_k$ can be obtained by marginalizing out the other variables from this CPT.

Let $\mathcal{C}_k$ be any clique containing a continuous random variable, say $\mathring{X}_k$. Let $g_k + 1$ be the domain size of the corresponding discretized random variable X_k. Finally, let $\mathcal{C}_j$ be the neighbor clique of $\mathcal{C}_k$. Then:

$$g_k(\mathring{x}_k|e) \propto \sum_{x_k=0}^{g} \mathcal{M}_{\mathcal{C}_j \to \mathcal{C}_k}(x_k) g_k(\mathring{x}_k|x_k) f_k(e_{\mathring{X}_k}|\mathring{x}_k) \tag{3.7}$$

is the posterior density of variable $\mathring{X}_k$.

Proof. [Proposition 3.3.1] By definition, the product of all the functions stored into junction tree $\mathcal{T}$ is a mixed probability distribution $P(x, \mathring{x}, e)$. So it satisfies the Shafer-Shenoy axioms [Shenoy and Shafer, 1990, Shafer, 1996]. As a consequence, the message-passing algorithm is sound and, for each clique, the function resulting from the multiplication of the functions stored in any clique by the messages sent to this clique is the joint (mixed) probability of the variables of the clique and evidence e. So, if the clique contains only discrete variables, after normalizing this resulting function, we necessarily get a joint posterior distribution of

the variables of the clique. On the other hand, if the clique contains a continuous variable $\mathring{X}_k$, then the resulting function is necessarily the posterior mixed distribution of X_k and $\mathring{X}_k$ given evidence e and, by marginalizing out X_k, we get the posterior density function of $\mathring{X}_k$. ■

As shown above, ctdBNs allow for the computation of the marginal *a posteriori* distributions of the continuous and discrete random variables. In addition, as shown in the next proposition, the algorithm proposed in this paper is very efficient for performing these computations. Notably, when the integrals of Equation (3.6) can be computed in O(1), the complexity of inference in ctdBN is exactly the same as that in classical discrete BNs. As a consequence, when tables for these integrals are available, like, e.g., when the ctdBN's conditional truncated densities are truncated normal distributions, inference in ctdBNs is as fast as that in discrete BNs.

Proposition 3.3.2 *Let w be the treewidth of $\mathcal{T}$ and let k denote the maximum domain size of the discrete and discretized random variables. Finally, let n be the number of random variables in the ctdBN and let $\overline{I}$ be the average complexity of computing one integral of Equation (3.6) (i.e., an integral for a given value of x_i) and $\overline{J}$ the average complexity of computing the product in Equation (3.7). Then the complexity of computing all the marginal posterior distributions of all the random variables is in $O(nk(k^w + \overline{I} + \overline{J}))$.*

Proof. [Proposition 3.3.2] There are at most n continuous random variables. Computing each message they send to their neighbor corresponds to perform $|\Omega_{X_i}| \leq k$ integrals. Hence the overall complexity of computing all these messages is in $O(nk\overline{I})$. As there are n random variables, there are at most n cliques containing only discrete variables. The complexity of computing their messages in both directions is therefore in $O(nk^{w+1})$. For the same reason, the complexity of sending messages from the cliques with only discrete variables to the cliques containing continuous variables is also $O(nk^{w+1})$. To compute the posterior of any discrete or discretized variable, it is sufficient to select one clique that contains it, to multiply the tables stored into this clique by all the messages sent to the clique and, then to marginalize out all the other variables. When performing the distribution phase, the tables stored into the clique are already multiplied by all the messages sent to the clique except one. So, to compute the posterior of the discrete variable, we just need to perform the last product required, with a complexity in $O(k^{w+1})$ and the summation (marginalizing-out) has the same complexity. Finally, there are n continuous variables. To compute the posterior density of a continuous variable, we must perform the operations of Equation (3.7). There are at most k products to perform and each product has an average complexity of $\overline{J}$, hence the overall

complexity of computing all the posterior densities is in $O(nk\overline{J})$. Overall, we get the complexity stated in the proposition. ■

Overall, inference in ctdBNs is fast because i) by construction, most of the inference's complexity lies in computations performed on discrete variables; and ii) whenever computations concern densities, either they correspond to compute a mixture of univariate conditional truncated densities (like in Equation (3.7)) or to compute the integral of a *univariate* function (like in Equation (3.6)).

3.4 Expressive power of ctdBNs: some experiments

In this section, we provide three sets of experiments. The first one is intended to assess the faithfulness and the inference speed of ctdBNs by comparing them to randomly generated hybrid Bayesian networks. The aim of the second set of experiments is to show the gain brought by ctdBNs over classical BNs. For this purpose, we illustrate the discrepancies between both models on classification problems derived from UCI datasets [Lichman, 2013]. Finally, the third set of experiments is devoted to the comparison between ctdBNs and MTBFs in order to highlight the inference scalability of ctdBNs compared to that of MTBFs.

3.4.1 Experiments on randomly-generated hybrid networks

For evaluating the expressive power of our model as well as the efficiency of the above inference algorithm, a set of hybrid Bayesian Networks (HBN) of different sizes are randomly generated [Moral et al., 2002]. In every HBN, half of the random variables are discrete; densities of the continuous variables are MTE potentials. The domain size of each discrete variable is randomly chosen between 2 and 4. The domain size $\Omega_{\mathring{X}_i}$ of each continuous random variable $\mathring{X}_i$ is randomly partitioned into 1, 2, or 3 subdomains Ω_k with probabilities of occurrence of 20%, 40% and 40% respectively. In each subdomain Ω_k, the number of exponential function terms in the density functions is chosen randomly among 0 (uniform distribution), 1 and 2, with respective probabilities 5%, 75% and 20%. In addition, the number of parents of all the nodes follow a Poisson distribution with a mean varying w.r.t. the number of nodes in the HBN as described in Table 3.1. For each number of nodes, 500 different HBNs are generated. Finally, the DAG structures are constrained to contain only one connected component. To construct these HBNs, we use the ELVIRA library (`http://leo.ugr.es/elvira`). Then, from each HBN, a dataset is generated, from which a ctdBN is learnt using the learning algorithm described in [Mabrouk et al., 2015], constraining it to use the same set of arcs as the HBN. As a consequence, the ctdBN can be considered as an approximation of an exact multivariate density function specified by the HBN.

$\#nodes$	$Mean(\#parents)$
4	2
8	1.83
16	1.66
32	1.5
64	1.33
128	1.16
256	1.1

Table 3.1: Average number of parents per node.

In each HBN, we perform an exact inference using the ELVIRA library in order to compute the marginal probabilities of all the random variables in the HBN. Similarly, for the ctdBN, we execute the inference algorithm of the preceding section, using a non-parallel implementation in C++ and the aGrUM library (`http://agrum.lip6.fr`). All the experiments are performed on a Linux box with an Intel Xeon at 2.40GHz and 128GB of RAM. For comparing MTEs and ctdBNs, we use two criteria:

1. The Jensen-Shannon Divergence (JSD) between the marginal distributions in the ctdBNs and those in the corresponding HBN; and

2. The times for computing these marginal distributions.

The first criterion allows to assess the expressive power of ctdBNs whereas the second one allows to assess the efficiency of our inference algorithm.

Tables 3.2 and 3.3 report for each network size, specified by its number of nodes, the average over the 500 networks generated for this size of the JSD between the marginal distributions of the nodes obtained in the ctdBN model and those obtained in the MTE model. The tables display the average of these JSDs over the nodes of the networks (μ_{JSD}) but also their standard deviations σ_{JSD} and their min ($\min_{JSD}$) and max ($\max_{JSD}$) values. As can be observed from these tables, the JSDs always remain small (remind that, for any distributions P, Q, $JSD(P\|Q) \in [0, 1]$). This shows that our model is expressive and faithful since its approximation of the true (MTE) densities is accurate. As shown in Table 3.4, which reports the inference execution times, this accuracy is not at the cost of inference performance: our algorithm significantly outperforms MTE inference and, the larger the network, the higher the discrepancy between the computation times of the two inference algorithms.

$\#nodes$	μ_{JSD}	σ_{JSD}	$\min_{JSD}$	$\max_{JSD}$
4	0.0065	0.0040	0.0024	0.0105
8	0.0076	0.0078	0.0007	0.0201
16	0.0078	0.0099	0.0003	0.0303
32	0.0122	0.0137	0.0004	0.0510
64	0.0354	0.0269	0.0016	0.1111
128	0.0831	0.0527	0.0028	0.2326
256	0.1329	0.0805	0.0033	0.3752

Table 3.2: JSD for discrete variables.

$\#nodes$	μ_{JSD}	σ_{JSD}	$\min_{JSD}$	$\max_{JSD}$
4	0.0059	0.0017	0.0042	0.0077
8	0.0064	0.0024	0.0036	0.0099
16	0.0064	0.0026	0.0030	0.0114
32	0.0071	0.0032	0.0028	0.0146
64	0.0101	0.0044	0.0033	0.0215
128	0.0161	0.0072	0.0045	0.0349
256	0.0222	0.0098	0.0054	0.0476

Table 3.3: JSD for continuous variables.

3.4.2 Comparisons with discrete BNs

In order to compare BNs and ctdBNs on real-world problems, we base our next experiments on the real-world datasets of the UCI repository [Lichman, 2013] reported in Table 3.5. In these datasets, all records with missing values are discarded. In each resulting dataset, there exists a discrete random variable, call it X_0, representing a classification variable. The other random variables can be either discrete (variables $\mathbf{X_D} = \{X_1, \ldots, X_d\}$) or continuous ($\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$). Our classification problem consists in estimating the most probable value of X_0 given some observation on the values of variables in $\mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$.

To address such a problem with Bayesian networks, we must first discretize all the continuous random variables. To do so, we exploit Friedman's discretization algorithm [Friedman and Goldszmidt, 1996]. After performing these discretizations, variables in $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$ are mapped into discretized variables in $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ and the mixed discrete/continuous dataset $\mathring{\mathcal{D}}$ of the UCI dataset is mapped into a fully discrete dataset $\mathcal{D}$. A BN $\mathcal{B}$ over $(X_0, X_1, \ldots, X_n)$ is then learnt from $\mathcal{D}$ using a hill climbing algorithm with an MDL score. To do so, we use the aGrUM library (`http://www.agrum.org`). This BN is then exploited for a classification task as follows: given some observation $e_{\mathring{X}_i}$ (resp.

$\#nodes$	T_{MTE}	T_{ctdBN}	T_{MTE}/T_{ctdBN}
4	40.92	0.27	151.56
8	279.16	0.84	332.33
16	7416.75	1.96	3784.06
32	42304.21	4.51	9380.09
64	88738.92	10.26	8649.02
128	94307.49	28.48	3311.36
256	122185.62	71.52	1708.41

Table 3.4: Inference computation times (in milliseconds).

dataset	#attributes	#classes	#instances	#continuous attr.
australian	14	2	690	6
cleve	14	2	296	13
crx	16	2	653	6
glass2	10	2	163	9
iris	5	3	150	4
pima	9	2	768	8
shuttle small	10	7	3866	9
vehicle	19	4	846	18

Table 3.5: UCI datasets used for BN/ctdBN comparisons in classification tasks.

e_{X_i}) on each continuous random variable $\mathring{X}_i$ (resp. discrete variable X_i), we enter belief $P(e_{\mathring{X}_i}|X_i)$ (resp. $P(e_{X_i}|X_i)$) into $\mathcal{B}$, so that the latter represents:

$$P(X_0,\ldots,X_n,e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n}) = \\ P(X_0|\mathbf{Pa}(X_0))\prod_{i=1}^{n}P(X_i|\mathbf{Pa}(X_i))\prod_{i=1}^{d}P(e_{X_i}|X_i)\prod_{i=d+1}^{n}P(e_{\mathring{X}_i}|X_i).$$

From this distribution, by means of a Shafer-Shenoy (exact) inference, we compute the posterior distribution $P(X_0|e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n})$ so that the most probable value for class variable X_0 is determined as:

$$x_0^* = \operatorname*{Argmax}_{X_0} P(X_0|e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n}).$$

To highlight the gain brought by ctdBNs over simple BNs, for each dataset, we construct our ctdBN as follows: we start from BN $\mathcal{B}$ computed in the preceding paragraphs and we add to it its respective conditional truncated densities $g_i(\mathring{X}_i|X_i)$, $i=d+1,\ldots,n$, defined as follows: let $\mathring{\Omega}_i^{obs}=\{\mathring{x}_{i,1},\mathring{x}_{i,2},\ldots,\mathring{x}_{i,N'}\}$ be the set of distinct observed values of $\mathring{X}_i$ in the dataset, sorted by increasing

order. The midpoints of $\mathring{\Omega}_i^{obs}$ are defined as:

$$m_{i,j} = \begin{cases} \mathring{x}_{i,1} - \frac{\mathring{x}_{i,2} - \mathring{x}_{i,1}}{2} & \text{if } j = 0, \\ \frac{\mathring{x}_{i,j} + \mathring{x}_{i,j+1}}{2} & \text{if } 1 \leq j < N', \\ \mathring{x}_{i,N'} + \frac{\mathring{x}_{i,N'} - \mathring{x}_{i,N'-1}}{2} & \text{if } j = N'. \end{cases}$$

Let $h_i : \Omega_{\mathring{X}_i} \mapsto \mathbb{R}$ be the histogram of $\mathring{X}_i$ whose bins correspond to intervals $[m_{i,j}, m_{i,j+1})$. Assume that $\mathring{X}_i$ has been discretized into X_i using cutpoints $\{t_i^1, \ldots, t_i^{g_i}\}$. Then we define conditional truncated densities $g_i(\mathring{X}_i | X_i = j)$, $j = 0, \ldots, g_i$, as the normalized histogram of h_i over $[t_i^j, t_i^{j+1})$, i.e.,

$$g_i(\mathring{x}_i | X_i = j) = \begin{cases} \dfrac{h_i(\mathring{x}_i)}{\int_{t_i^j}^{t_i^{j+1}} h_i(\mathring{x}) d\mathring{x}} & \text{if } \mathring{x}_i \in [t_i^j, t_i^{j+1}), \\ 0 & \text{otherwise.} \end{cases}$$

The ctdBN therefore represents the following mixed probability distribution:

$$g(X_0, \ldots, X_n, \mathring{X}_{d+1}, \ldots, \mathring{X}_n) = P(X_0 | \mathbf{Pa}(X_0)) \prod_{i=1}^{n} P(X_i | \mathbf{Pa}(X_i)) \prod_{i=d+1}^{n} g_i(\mathring{X}_i | X_i).$$

The same evidence e_{X_i} and $e_{\mathring{X}_i}$ as those of the BN are entered into the ctdBN. However, the latter are included into the ctdBN as beliefs $f_i(e_{\mathring{X}_i} | \mathring{X}_i)$ as ctdBNs can cope with more precise evidence than mere beliefs $P(e_{\mathring{X}_i} | X_i)$ about discretized random variables X_i. Therefore, after entering evidence, the ctdBN represents:

$$g(X_0, \ldots, X_n, \mathring{X}_{d+1}, \ldots, \mathring{X}_n, e_{X_1}, \ldots, e_{X_d}, e_{\mathring{X}_{d+1}}, \ldots, e_{\mathring{X}_n}) = P(X_0 | \mathbf{Pa}(X_0)) \prod_{i=1}^{n} P(X_i | \mathbf{Pa}(X_i)) \prod_{i=1}^{d} P(e_{X_i} | X_i) \prod_{i=d+1}^{n} g_i(\mathring{X}_i | X_i) f_i(e_{\mathring{X}_i} | \mathring{X}_i).$$

From this distribution, using the algorithm provided in Section 3.3, we compute $g(X_0 | e_{X_1}, \ldots, e_{X_d}, e_{\mathring{X}_{d+1}}, \ldots, e_{\mathring{X}_n})$ and the most probable value for class variable X_0 is $x_0^* = \text{Argmax}_{X_0} g(X_0 | e_{X_1}, \ldots, e_{X_d}, e_{\mathring{X}_{d+1}}, \ldots, e_{\mathring{X}_n})$.

Finally, to perform our experiments, each dataset of Table 3.5 is randomly shuffled 100 times. Each resulting dataset is splitted into a learning set (70%) and a test set (30%). So, overall, for each UCI dataset, 100 different learning sets and their respective 100 test sets are created. From each learning set, we learn a BN and a ctdBN and, then, for each record of the corresponding test set, we estimate the most probable values of class variable X_0 given observations on

$X_1, \ldots, X_d, \mathring{X}_{d+1}, \ldots, \mathring{X}_n$ according to these two models. These estimations are then compared with the true values of X_0 observed in the test set and the accuracy of the model (BN,ctdBN) is defined as the proportion of correct estimations performed. The latter depend on the kind of observations available, so we shall now describe those used in the experiments.

First, all observations over discrete variables are supposed to be precise. Now, assume that observation $e_{\mathring{X}_i}$ over continuous variable $\mathring{X}_i$ is also precise, i.e., it is equal to the observed value of $\mathring{X}_i$ in the record. Then, for the ctdBN, $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ is a Dirac function, which means that message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$, as defined in Eq. (3.6), is a zero filled vector, bringing no information. In addition, the BN is unable to handle such precise information. What can be handled by the BN is the (less precise) observation that "the discretized value of the observed value of $\mathring{X}_i$ is equal to j". Such information is exactly equivalent to "$\mathring{X}_i$ belongs to interval $[t_i^j, t_i^{j+1})$", where $[t_i^j, t_i^{j+1})$ is the discretization interval containing the observed value of $\mathring{X}_i$. In this case, $P(e_{\mathring{X}_i}|X_i)$, is a vector filled with zeros except for a 1 in the cell corresponding to the discretized value of $\mathring{X}_i$. If we enter the same information into the ctdBN, Message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ is exactly proportional to $P(e_{\mathring{X}_i}|X_i)$ and, therefore, the BN and the ctdBN provide the same estimation for X_0.

The advantage of ctdBNs over standard BNs becomes visible when observations are imprecise. So, in our experiments, all the continuous variables $\mathring{X}_i$ are imprecisely observed and the belief $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ is always expressed as a normal distribution centered on the observed value of $\mathring{X}_i$. For the BN, vector $P(e_{\mathring{X}_i}|X_i)$ can then simply be computed as:

$$P(e_{\mathring{X}_i}|X_i = j) = \int_{t_i^j}^{t_i^{j+1}} f_i(e_{\mathring{X}_i}|\mathring{x}_i)\, d\mathring{x}_i \quad \text{for all } j. \tag{3.8}$$

When the standard deviations of the normal distributions are sufficiently small, in the BN, $P(e_{\mathring{X}_i}|X_i)$ is approximately equal to a vector filled with zeros except for one cell equal to 1 and, in the ctdBN, Message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ of Eq. (3.6) is approximately proportional to $P(e_{\mathring{X}_i}|X_i)$. So, both models are equivalent. However, when standard deviations are higher, i.e., when observations are less precise, $P(e_{\mathring{X}_i}|X_i)$ can contain several non-zero cells and, from Eq. (3.6), it is clear that Message $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ can differ from $P(e_{\mathring{X}_i}|X_i)$ and bring more refined information than $P(e_{\mathring{X}_i}|X_i)$. In our experiments, all the standard deviations of the continuous variables are kept sufficiently small that $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j} \propto P(e_{\mathring{X}_i}|X_i)$, except for the few variables mentioned in Table 3.6 in which the standard deviations displayed introduce discrepancies between $\mathcal{M}_{\mathcal{C}_i \to \mathcal{C}_j}$ and $P(e_{\mathring{X}_i}|X_i)$. Note that these standard deviations are often much smaller than the range of the random variable.

With the observations as described above, the average accuracies for the different UCI datasets over the 100 corresponding test sets, as well as their standard

Dataset	standard deviations and variables' domain sizes
australian	$\mathring{X}_7 : 19\ (28.5)$, $\mathring{X}_{10} : 27\ (68)$, $\mathring{X}_{13} : 98\ (2000)$, $\mathring{X}_{14} : 83.5\ (100000)$
cleve	$\mathring{X}_8 : 21\ (133)$
crx	$\mathring{X}_{11} : 37\ (68)$, $\mathring{X}_{15} : 235\ (100000)$
glass	$\mathring{X}_3 : 0.55\ (4.5)$, $\mathring{X}_4 : 0.1\ (3.8)$, $\mathring{X}_5 : 0.6\ (5.6)$, $\mathring{X}_6 : 0.55\ (6.2)$
	$\mathring{X}_8 : 0.35\ (3.15)$
iris	$\mathring{X}_3 : 0.03\ (6)$, $\mathring{X}_4 : 0.085\ (2)$
pima	$\mathring{X}_1 : 6.6\ (18)$, $\mathring{X}_2 : 15.4\ (200)$, $\mathring{X}_4 : 0.85\ (100)$, $\mathring{X}_6 : 1.3\ (67)$
shuttle_small	$\mathring{X}_1 : 3.5\ (74)$
vehicle	$\mathring{X}_6 : 3\ (54)$, $\mathring{X}_{11} : 34\ (191)$

Table 3.6: The standard deviations for the beliefs on the observations of the $\mathring{X}_i$'s. The domain sizes of the $\mathring{X}_i$'s are given inside parentheses. In each dataset, the first variable/column is called X_1 or $\mathring{X}_1$, the second one X_2 or $\mathring{X}_2$, etc. Class variable X_0 is always located in the last column of the dataset.

deviations, are reported in Table 3.7. From this table, it is clear that ctdBNs outperform BNs for classification tasks. The way we constructed the ctdBNs from the BNs, this improvement is necessarily due to the conditional truncated densities contained in the ctdBNs.

Dataset	% BN Acc.	% ctdBN Acc.	Gain Acc.	p-value
australian	84.36 ± 3.08	$\mathbf{85.72 \pm 2.12}$	1.34 ± 2.45	0.2912
cleve	82.89 ± 3.67	$\mathbf{83.22 \pm 3.50}$	0.34 ± 0.96	0.3632
crx	85.36 ± 2.48	$\mathbf{86.38 \pm 2.08}$	1.02 ± 2.12	0.3156
glass	89.94 ± 3.56	$\mathbf{91.88 \pm 2.98}$	1.94 ± 2.61	0.2236
iris	94.73 ± 2.62	$\mathbf{95.49 \pm 2.82}$	0.76 ± 1.45	0.3015
pima	73.64 ± 2.67	$\mathbf{74.37 \pm 2.69}$	0.74 ± 1.18	0.2676
shuttle small	84.92 ± 5.93	$\mathbf{92.66 \pm 3.41}$	7.74 ± 4.21	0.0336
vehicle	35.63 ± 6.02	$\mathbf{52.21 \pm 7.58}$	16.57 ± 7.97	0.0000

Table 3.7: Comparisons between BNs and ctdBNs for classification tasks. The first two columns present the average accuracies and the accuracies' standard deviations over the 100 datasets constructed from each UCI dataset. The third column displays the gain in accuracy of using ctdBNs instead of BNs (the subtraction of the first column from the second one). Assuming that accuracies are distributed w.r.t. normal distributions of parameters the average BN accuracy and the variance of the BN accuracy, the last column shows the p-value obtained by the ctdBN accuracy.

3.4.3 Comparisons with MTBFs

In this subsection, we highlight the faithfulness of ctdBNs by comparing them with MTBFs, more precisely with MOPs [Shenoy, 2011] and MTEs [Moral et al., 2001]. We show that ctdBNs can approximate effectively these MTBFs while being more efficient in terms of inference. For this purpose, we generate MTBFs $\Phi(\mathring{X}_1, \ldots, \mathring{X}_n)$ as products of bivariate MTBF potentials:

$$\Phi(\mathring{X}_1, \ldots, \mathring{X}_n) = \Phi_1(\mathring{X}_1, \mathring{X}_2) \times \Phi_2(\mathring{X}_2, \mathring{X}_3) \times \cdots \times \Phi_{n-1}(\mathring{X}_{n-1}, \mathring{X}_n).$$

This decomposition has been chosen in order to make inference in MTBFs as fast as possible. Indeed, the corresponding junction tree contains only cliques of size two, having at most two neighbors, which limits the combinatorics of the algebraic operations performed during inference. All these MTBF potentials are defined over some continuous variables $\mathring{X}_i$ whose domain is $\Omega_{\mathring{X}_i} = [0, 10]$. To make MTEs as efficient as possible for inference, we express MTE potentials $\Phi_i^{MTE}(\mathring{X}_i, \mathring{X}_{i+1})$ by only one exponential term:

$$\Phi_i^{MTE}(\mathring{X}_i, \mathring{X}_{i+1}) = a_{i,0} + a_{i,1} \exp\left(b_{i,0}\mathring{X}_i + b_{i,1}\mathring{X}_{i+1}\right),$$

with $a_{i,0}, a_{i,1} \in [0, 1]$ and $b_{i,0}, b_{i,1} \in [0.5, 1]$ chosen randomly.

In order to make the shape of MOP potentials not easily captured by affine distributions[1] (which we use in our ctdBNs) while guaranteeing that inferences in MOPs are as fast as possible, we define MOP potentials as polynomials of degree 6, more precisely as polynomials of degree 3 in each of their variables, i.e.,:

$$\Phi_i^{MOP}(\mathring{X}_i, \mathring{X}_{i+1}) = \sum_{j=0}^{3}\sum_{k=0}^{3} c_{i,j,k}\mathring{X}_i^j\mathring{X}_{i+1}^k,$$

with $c_{i,j,k}$ chosen randomly in interval $[0, 1]$.

Our goal is to approximate these MTBFs by ctdBNs $\mathcal{B}$ encoding a mixed probability distribution $g(X_1, \ldots, X_n, \mathring{X}_1, \ldots, \mathring{X}_n)$. To construct $\mathcal{B}$, we first discretize every continuous variable $\mathring{X}_i$ in 50 equally-sized intervals, hence resulting in discretized random variables X_i. Then, we construct a discrete BN $\mathcal{B}_d$ over $X_1, \ldots, X_n$ whose independence structure corresponds to that of the MTBF, i.e., to the structure shown in Figure 3.3. Finally, ctdBN $\mathcal{B}$ is defined as $\mathcal{B}_d$ to which are added conditional truncated densities $g_i(\mathring{X}_i|X_i)$. Therefore, the ctdBN encodes the following mixed distribution:

$$g(X_1, \ldots, X_n, \mathring{X}_1, \ldots, \mathring{X}_n) = P_{\mathcal{B}_d}(X_1)\prod_{i=2}^{n} P_{\mathcal{B}_d}(X_i|X_{i-1})\prod_{i=1}^{n} g_i(\mathring{X}_i|X_i).$$

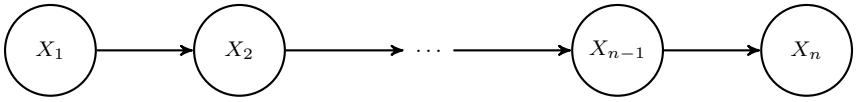


Figure 3.3: The BN structure used for approximating MTBFs.

As a result, the junction tree used for inferences with ctdBN $\mathcal{B}$ always contains only cliques of size two, exactly like that of the MTBF. Since inference complexity with junction trees is exponential in the treewidth, imposing the structure of Figure 3.3 allows to perform a fair comparison of how well ctdBNs' and MTBFs' inferences scale with increasing numbers of random variables.

As the structure of $\mathcal{B}$ is imposed, only $\mathcal{B}$'s parameters (CPTs $P_{\mathcal{B}_d}(X_i|X_{i-1})$ and functions $g_i(\mathring{X}_i|X_i)$) need be learnt. This learning is performed iteratively, i.e., in a first step, functions $P_{\mathcal{B}_d}(X_1)$, $P_{\mathcal{B}_d}(X_2|X_1)$, $g_i(\mathring{X}_1|X_1)$ and $g_i(\mathring{X}_2|X_2)$ are determined. Then, assuming that ctdBN $\mathcal{B}$ has been constructed for variables $X_1, \mathring{X}_1$ up to $X_{i-1}, \mathring{X}_{i-1}$, we learn $P_{\mathcal{B}_d}(X_i|X_{i-1})$ and $g_i(\mathring{X}_i|X_i)$. More precisely, to learn the parameters related to $X_i, \mathring{X}_i$, we first perform an inference in the MTBF in order to compute:

$$\Phi(\mathring{X}_{i-1}, \mathring{X}_i) = \sum_{\{\mathring{X}_1,\ldots,\mathring{X}_n\}\setminus\{\mathring{X}_{i-1},\mathring{X}_i\}} \Phi(\mathring{X}_1, \ldots, \mathring{X}_n).$$

We also compute $\Phi(\mathring{X}_i) = \sum_{\mathring{X}_{i-1}} \Phi(\mathring{X}_{i-1}, \mathring{X}_i)$. Then we sample 500000 times the potentials $\Phi(\mathring{X}_{i-1}, \mathring{X}_i)$ and $\Phi(\mathring{X}_i)$ using the Metropolis-Hastings algorithm [Hastings, 1970] and the Inverse Transform Sampling's method, respectively. Discretizing these samples using the discretizations of $\mathring{X}_{i-1}$ and $\mathring{X}_i$ defined above results in new discrete samples from which we determine by maximum likelihood some probability distributions $P(X_{i-1}|X_i)$ and $P(X_i)$ respectively. Note that, due to the finite size of the samples, the estimated distributions $P(X_{i-1}|X_i)$ and $P(X_i)$ are not necessarily equal to $P_{\mathcal{B}_d}(X_{i-1}|X_i)$ and $P_{\mathcal{B}_d}(X_i)$. From these two distributions, we compute $P(X_{i-1}, X_i) = P(X_{i-1}|X_i) \times P(X_i)$. The goal of constructing joint distribution $P(X_{i-1}, X_i)$ in this two-step process rather than directly from the sample of $\Phi(\mathring{X}_{i-1}, \mathring{X}_i)$ is to ensure that, in this joint, the marginal distribution $P(X_i)$ is as close as possible to distribution $\Phi(\mathring{X}_i)$, which may not be the case when sampling with Metropolis-Hastings uniquely on pairs $(\mathring{X}_{i-1}, \mathring{X}_i)$, due to the finite size of the samples. Finally, we define $P_{\mathcal{B}_d}(X_1) = P(X_1)$ and, for every $i > 1$, $P_{\mathcal{B}_d}(X_i|X_{i-1}) = P(X_{i-1}, X_i)/P_{\mathcal{B}_d}(X_{i-1})$, where $P_{\mathcal{B}_d}(X_{i-1})$ is computed by inference in the ctdBN constructed so far.

[1]The conditional truncated densities we use in our ctdBNs are in fact Beta rectangular distributions with parameters $\alpha = 2$ and $\beta = 1$.

In our experiments, for every j, conditional density $g_i(\mathring{X}_i|X_i = j)$ is an affine function on discretization interval $[t_i^j, t_i^{j+1})$ and is equal to 0 everywhere else. Therefore, it is of the form $g_i(\mathring{x}_i|X_i = j) = \alpha_{i,j}\mathring{x}_i + \beta_{i,j}$ on this interval and, since this is a probability density function, we have that:

$$\int_{t_i^j}^{t_i^{j+1}} g_i(\mathring{x}_i|X_i = j)d\mathring{x}_i = 1.$$

As a consequence, the following equation holds for all $\mathring{x}_i \in \Omega_{\mathring{X}_i}$:

$$g_i(\mathring{x}_i|X_i = j) = \alpha_{i,j}\left(\mathring{x}_i - \frac{t_i^j + t_i^{j+1}}{2}\right) + \frac{1}{t_i^{j+1} - t_i^j}.$$

Using the projection over $\mathring{X}_i$ of the 500000-record sample of $\Phi(\mathring{X}_{i-1}, \mathring{X}_i)$ previously used for estimating $P_{\mathcal{B}_d}(X_i|X_{i-1})$, we can now estimate $\alpha_{i,j}$ by maximum likelihood under the constraint that $g_i(\cdot)$ is non-negative on $[t_i^j, t_i^{j+1})$. As there is no closed-form formula for the optimal value of $\alpha_{i,j}$, we solve this constrained optimization problem by the Newton-Raphson method.

The experiments are conducted as follows: we generate MTBFs (both MTEs and MOPs) with $n = 4$, 8, 16, 32, 64, 128 and 256 variables. For each number of variables, 25 MTEs and 25 MOPs are generated. Exact inferences are performed in all these models using Variable Elimination [Dechter, 1999] in order to determine the distribution $\Phi(\mathring{X}_n)$ of the last random variable. For each MOP and each MTE, we also learn a ctdBN as described above and execute the inference algorithm described in Section 3.3 to determine the distribution $g(\mathring{X}_n)$ of $\mathring{X}_n$.

The inference times are reported in Tables 3.8 and 3.9. They highlight the scalability of ctdBNs. The ratios of the average inference time by the number of variables, i.e., the last two columns of the tables, are displayed in Figures 3.4

n	$T_{mte}(ms)$	$T_{ctdBN}(ms)$	T_{mte}/n	T_{ctdBN}/n
4	0.30 ± 0.03	1.59 ± 0.16	0.07	0.40
8	0.57 ± 0.08	3.53 ± 0.31	0.07	0.44
16	1.67 ± 1.10	7.14 ± 0.41	0.10	0.45
32	5.79 ± 0.87	13.99 ± 1.35	0.18	0.44
64	23.68 ± 2.01	29.22 ± 1.92	0.37	0.46
128	122.37 ± 10.99	50.87 ± 6.88	0.96	0.40
256	708.11 ± 37.12	96.02 ± 13.79	2.77	0.38

Table 3.8: Average inference times (plus standard deviations) for MTEs and ctdBNs. These averages are computed over the 25 different networks defined for each number n of variables.

n	$T_{mop}(ms)$	$T_{ctdBN}(ms)$	T_{mop}/n	T_{ctdBN}/n
4	0.79 ± 0.16	1.59 ± 0.19	0.20	0.40
8	1.76 ± 0.29	3.40 ± 0.26	0.22	0.42
16	5.04 ± 0.81	6.94 ± 0.78	0.32	0.43
32	16.41 ± 1.42	14.40 ± 0.89	0.51	0.45
64	50.96 ± 6.89	29.21 ± 1.55	0.80	0.46
128	217.07 ± 22.04	60.19 ± 4.87	1.70	0.47
256	1063.48 ± 51.82	111.34 ± 16.88	4.15	0.43

Table 3.9: Average inference times (plus standard deviations) for MOPs and ctdBNs. These averages are computed over the 25 different networks defined for each number n of variables.

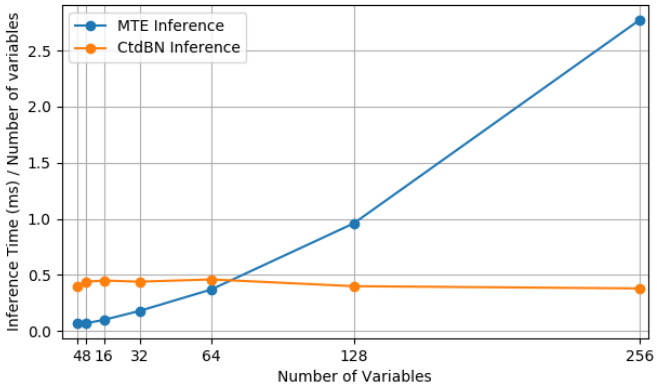


Figure 3.4: The ratio of inference times in MTEs and ctdBNs by the number of variables.

and 3.5. It is clear that the MTBFs' inference times increase exponentially with the number of variables, even though the largest clique always remains of size 2. This results from the multiplications of the algebraic functions performed during the inferences that tend to produce new functions with an ever increasing number of parameters. Even marginalizations cannot restrain this increase. Unlike in MTBFs, in ctdBNs, the number of operations performed on each clique remains always the same during inference. This explains the linear increase in computation times when the number of variables increase. This also corroborates the inference complexity provided in Proposition 3.3.2.

Of course, better inference times are attractive only if the estimations by ctdBNs approximate pretty well those of MTBFs. As a first hint that this is the case, Table 3.10 displays the average Jensen-Shannon Divergence (JSD) between the distributions $\Phi(\mathring{X}_n)$ and $g(\mathring{X}_n)$ computed previously. The low JSD values show

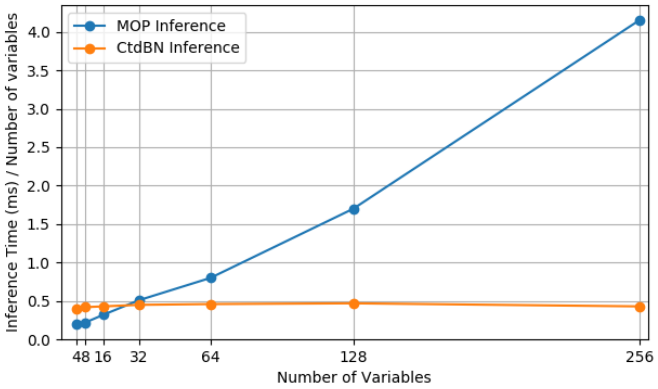


Figure 3.5: The ratio of inference times in MOPs and ctdBNs by the number of variables.

that ctdBNs approximate effectively MTBFs. To precise these results, we add some small noise to distributions $\Phi(\mathring{X}_n)$ computed previously, hence resulting in new distributions $\Psi(\mathring{X}_n)$. Comparing the JSDs between $\Phi(\mathring{X}_n)$ and $\Psi(\mathring{X}_n)$ on one hand and between $\Phi(\mathring{X}_n)$ and $g(\mathring{X}_n)$ on the other, we show that ctdBNs provide better approximations than the slightly perturbed distributions, hence highlighting the ctdBN's faithfulness, while enabling much faster inferences than MTBFs, as shown above.

For the MTEs, the marginal distribution inferred for $\mathring{X}_n$ is of the form $\Phi^{MTE}(\mathring{X}_n) = a_0 + b_0 \exp(b_1 \mathring{X}_n)$. We perturb its parameters by ϵ, for different values of ϵ, as follows:

$$\Psi_{\epsilon}^{MTE}(\mathring{X}_n) \propto (1+\epsilon)a_0 + (1-\epsilon)b_0 \exp\left((1+\epsilon)b_1 \mathring{X}_n\right).$$

Note that $\Psi_{\epsilon}^{MTE}(\mathring{X}_n)$ is not strictly equal to the right hand side of the above equation, but only proportional to it, so that the integral of Ψ_{ϵ}^{MTE} over $\Omega_{\mathring{X}_n}$ is equal to one, hence ensuring that Ψ_{ϵ}^{MTE} is a probability density function. For MOPs, the marginal distribution of $\mathring{X}_n$ is of the form $\Phi^{MOP}(\mathring{X}_n) = c_0 + c_1 \mathring{X}_n + c_2 \mathring{X}_n^2 + c_3 \mathring{X}_n^3$. We perturb it up to ϵ as follows:

$$\Psi_{\epsilon}^{MOP}(\mathring{X}_n) \propto (1+\epsilon)c_0 + (1-\epsilon)c_1 \mathring{X}_n^{1+\epsilon} + (1+\epsilon)c_2 \mathring{X}_n^{2(1-\epsilon)} + (1-\epsilon)c_3 \mathring{X}_n^{3(1+\epsilon)}.$$

As for Ψ_{ϵ}^{MTE}, function Ψ_{ϵ}^{MOP} is normalized so that its integral over $\Omega_{\mathring{X}_n}$ is equal to one. The average JSDs between distributions $\Phi(\mathring{X}_n)$ and $\Psi_{\epsilon}(\mathring{X}_n)$ for $\epsilon = 0.1$, 0.05 and 0.01 are provided in Tables 3.11 and 3.12 for MTEs and MOPs respectively. As can be observed, for $\epsilon = 0.05$, both for MTEs and MOPs, and whatever the number of random variables, the JSDs between the true distribution $\Phi(\mathring{X}_n)$ and

n	$JSD[\Phi^{MTE}(\mathring{X}_n);\ g(\mathring{X}_n)]$	$JSD[\Phi^{MOP}(\mathring{X}_n);\ g(\mathring{X}_n)]$
4	$2.47 \times 10^{-4} \pm 7.19 \times 10^{-5}$	$8.07 \times 10^{-5} \pm 7.40 \times 10^{-6}$
8	$2.33 \times 10^{-4} \pm 8.28 \times 10^{-5}$	$8.33 \times 10^{-5} \pm 4.24 \times 10^{-6}$
16	$2.32 \times 10^{-4} \pm 7.56 \times 10^{-5}$	$8.24 \times 10^{-5} \pm 6.13 \times 10^{-6}$
32	$2.56 \times 10^{-4} \pm 8.15 \times 10^{-5}$	$8.30 \times 10^{-5} \pm 5.48 \times 10^{-6}$
64	$2.58 \times 10^{-4} \pm 9.05 \times 10^{-5}$	$8.10 \times 10^{-5} \pm 7.11 \times 10^{-6}$
128	$2.07 \times 10^{-4} \pm 6.72 \times 10^{-5}$	$7.85 \times 10^{-5} \pm 7.83 \times 10^{-6}$
256	$2.39 \times 10^{-4} \pm 7.87 \times 10^{-5}$	$8.18 \times 10^{-5} \pm 5.11 \times 10^{-6}$

Table 3.10: Average Jensen-Shannon Divergences between $\Phi(\mathring{X}_n)$ and $g(\mathring{X}_n)$ for different numbers of variables.

	$JSD[\Phi^{MTE}(\mathring{X}_n);\ \Psi^{MTE}(\mathring{X}_n)]$		
n	$\epsilon = 0.1$	$\epsilon = 0.05$	$\epsilon = 0.01$
4	$1.09 \times 10^{-3} \pm 4.02 \times 10^{-5}$	$2.84 \times 10^{-4} \pm 1.14 \times 10^{-5}$	$1.18 \times 10^{-5} \pm 5.06 \times 10^{-7}$
8	$1.08 \times 10^{-3} \pm 5.39 \times 10^{-5}$	$2.81 \times 10^{-4} \pm 1.51 \times 10^{-5}$	$1.16 \times 10^{-5} \pm 6.64 \times 10^{-7}$
16	$1.11 \times 10^{-3} \pm 1.80 \times 10^{-4}$	$2.90 \times 10^{-4} \pm 5.12 \times 10^{-5}$	$1.20 \times 10^{-5} \pm 2.27 \times 10^{-6}$
32	$1.08 \times 10^{-3} \pm 5.32 \times 10^{-5}$	$2.83 \times 10^{-4} \pm 1.50 \times 10^{-5}$	$1.17 \times 10^{-5} \pm 6.58 \times 10^{-7}$
64	$1.08 \times 10^{-3} \pm 4.58 \times 10^{-5}$	$2.83 \times 10^{-4} \pm 1.30 \times 10^{-5}$	$1.17 \times 10^{-5} \pm 5.77 \times 10^{-7}$
128	$1.06 \times 10^{-3} \pm 4.89 \times 10^{-5}$	$2.77 \times 10^{-4} \pm 1.37 \times 10^{-5}$	$1.15 \times 10^{-5} \pm 6.04 \times 10^{-7}$
256	$1.12 \times 10^{-3} \pm 1.40 \times 10^{-4}$	$2.93 \times 10^{-4} \pm 3.87 \times 10^{-5}$	$1.21 \times 10^{-5} \pm 1.67 \times 10^{-6}$

Table 3.11: Average Jensen-Shannon Divergences between $\Phi^{MTE}(\mathring{X}_n)$ inferred by MTE and its perturbed distributions.

the distribution $g(\mathring{X}_n)$ inferred by ctdBN are smaller than those between $\Phi(\mathring{X}_n)$ and ϵ-perturbed distributions $\Psi_\epsilon(\mathring{X}_n)$. This supports the fact that ctdBNs approximate very well MTBFs. Indeed, in real-world applications, the parameters of MTEs and MOPs are learnt from datasets and perturbed probability density functions $\Psi_\epsilon(\mathring{X}_n)$ can be seen as the result of the imprecision on the values of these parameters due to this learning.

	$JSD[\Phi^{MOP}(\mathring{X}_n);\ \Psi^{MOP}(\mathring{X}_n)]$		
n	$\epsilon = 0.1$	$\epsilon = 0.05$	$\epsilon = 0.01$
4	$8.86 \times 10^{-4} \pm 1.37 \times 10^{-4}$	$2.31 \times 10^{-4} \pm 3.58 \times 10^{-5}$	$9.54 \times 10^{-6} \pm 1.48 \times 10^{-6}$
8	$8.17 \times 10^{-4} \pm 7.32 \times 10^{-5}$	$2.13 \times 10^{-4} \pm 1.92 \times 10^{-5}$	$8.80 \times 10^{-6} \pm 7.93 \times 10^{-7}$
16	$8.43 \times 10^{-4} \pm 8.58 \times 10^{-5}$	$2.19 \times 10^{-4} \pm 2.16 \times 10^{-5}$	$9.04 \times 10^{-6} \pm 8.65 \times 10^{-7}$
32	$8.33 \times 10^{-4} \pm 7.70 \times 10^{-5}$	$2.17 \times 10^{-4} \pm 1.98 \times 10^{-5}$	$8.96 \times 10^{-6} \pm 8.10 \times 10^{-7}$
64	$8.94 \times 10^{-4} \pm 1.17 \times 10^{-4}$	$2.33 \times 10^{-4} \pm 3.04 \times 10^{-5}$	$9.61 \times 10^{-6} \pm 1.25 \times 10^{-6}$
128	$9.24 \times 10^{-4} \pm 1.54 \times 10^{-4}$	$2.40 \times 10^{-4} \pm 3.91 \times 10^{-5}$	$9.88 \times 10^{-6} \pm 1.57 \times 10^{-6}$
256	$8.59 \times 10^{-4} \pm 1.06 \times 10^{-4}$	$2.24 \times 10^{-4} \pm 2.76 \times 10^{-5}$	$9.25 \times 10^{-6} \pm 1.14 \times 10^{-6}$

Table 3.12: Jensen-Shannon Divergences for marginalized distributions of $\mathring{X}_n$ in MOPs w.r.t. the marginals obtained using perturbed MOPs.

We can therefore conclude that ctdBNs outperform MTBFs in terms of scalability of inference. As shown above, the cost of this speed increase is a slight imprecision in the results of the inferences. All experiments have been performed using the C++ aGrUM library (`http://www.agrum.org`) on a Linux box

with an Intel Xeon at 2.40GHz and 128GB of RAM.

3.5 Conclusion

In this chapter, ctdBNs, a new graphical model for handling uncertainty over sets of continuous and discrete variables, have been introduced. We have proved that ctdBNs can approximate (arbitrarily well) any Lipschitz mixed probability distribution. So, theoretically, most of the mixed probability distributions used in real-world situations can be approximated by ctdBNs. Experiments highlight that this result is not only theoretic: in practice, ctdBNs are very expressive and can be exploited efficiently for diagnosis and classification tasks. A junction tree-based inference algorithm has also been provided. Its theoretical computational complexity has been given and it shows that inference in ctdBNs is essentially similar to that in classical discrete BNs. Here again, the experiments provided at the end of the chapter highlight the tractability of inference in practical situations.

Chapter 4

Conditional Densities Bayesian Networks

In the preceding chapter, we proposed a new model called Conditional Truncated Bayesian networks. This one is attractive because it is almost as expressive as MTE and its inference engine is as fast as that of CLG models. However, it has some drawbacks, notably in terms of structure learning because the combination of the discretizations and the conditional truncated densities it relies on may induce some side effects that prevent using classical scoring functions. This issue is not only related to ctdBNs but also to any algorithm in which both the discretization of the continuous random variables and the structure are learnt at the same time. The problem is the following: scoring functions represent probability $p(\mathcal{G}|\mathring{\mathcal{D}})$, where $\mathring{\mathcal{D}}$ denotes the database from which the model is learnt and $\mathcal{G}$ is the structure of the model. In this context, learning the best structure consists of determining $\mathcal{G}^* = \text{Argmax}_{\mathcal{G}}\, p(\mathcal{G}|\mathring{\mathcal{D}})$ and, using Bayes rule, we have that $\mathcal{G}^* = \text{Argmax}_{\mathcal{G}}\, p(\mathring{\mathcal{D}}|\mathcal{G})p(\mathcal{G})$. Term $p(\mathcal{G})$ is a prior over the possible structures and does not raise any issue. The problem comes from likelihood $p(\mathring{\mathcal{D}}|\mathcal{G})$. Assuming that all the records in Database $\mathring{\mathcal{D}}$ are i.i.d., we have that:

$$p(\mathring{\mathcal{D}}|\mathcal{G}) = \int_{\boldsymbol{\theta}} \prod_{j=1}^{N} p(\mathring{\mathbf{x}}^{(j)}|\mathcal{G}, \boldsymbol{\theta})\pi(\boldsymbol{\theta}|\mathcal{G})d\boldsymbol{\theta},$$

where N denotes the number of records in the database and $\mathring{\mathbf{x}}^{(j)}$ corresponds to the jth record and $\pi(\boldsymbol{\theta}|\mathcal{G})$ is the prior over parameters $\boldsymbol{\theta}$. Now, exploiting the decomposition of p w.r.t. $\mathcal{G}$, the score corresponding structure $\mathcal{G}$ of a ctdBN would be:

$$p(\mathring{\mathcal{D}}|\mathcal{G}) = \int_{\boldsymbol{\theta}} \prod_{j=1}^{N} \prod_{i=1}^{n} P(x_i^{(j)}|\mathbf{Pa}(x_i^{(j)}), \boldsymbol{\theta}) \prod_{i=d+1}^{n} f(\mathring{x}_i^{(j)}|x_i^{(j)}, \boldsymbol{\theta})\pi(\boldsymbol{\theta}|\mathcal{G})\, d\boldsymbol{\theta}, \quad (4.1)$$

where x_i, $i = d+1, \ldots, n$, is the discretized counterpart of variable $\mathring{x}_i$. Now, when the discretizations of the continuous variables and the structure of the model are learnt at the same time, in order to maximize Eq. (4.1), it is sufficient to choose any continuous variable, say $\mathring{X}_{i_0}$, to discretize it arbitrarily, except for one interval of discretization $[t_k, t_{k+1})$ made arbitrarily small but still containing some $\mathring{x}_{i_0}^{(j)}$ of the database. By definition, f is a truncated density, so $\int_{t_k}^{t_{k+1}} f(\mathring{x}|x_{i_0}^{(j)}, \boldsymbol{\theta}) d\mathring{x} = 1$. As interval $[t_k, t_{k+1})$ is arbitrarily small, the integral can only be equal to one if density $f(\mathring{x}|x_{i_0}^{(j)}, \boldsymbol{\theta})$ is arbitrarily high. As a consequence, whatever Structure $\mathcal{G}$, the above discretization of $\mathring{X}_{i_0}$ implies that $p(\mathring{\mathcal{D}}|\mathcal{G})$ tends toward $+\infty$. This feature prevents any scoring function representing $p(\mathcal{G}|\mathring{\mathcal{D}})$ to be used for learning. This explains why scoring functions like the one provided in [Monti and Cooper, 1998] face serious issues in practice. In [Friedman and Goldszmidt, 1996], the authors avoid this issue by adding to their MDL score the entropy-based term named $DL_{\mathcal{F}}$ (see Eq.(2.14) which, when examined closely, does not fit very well with the MDL criterion: indeed, the information it encodes can be significantly compressed using other encoding schemes. But this term counteracts the impact of $f(\mathring{x}_{i_0}^{(j)}|x_{i_0}^{(j)}, \boldsymbol{\theta})$ and makes the discretizations tend toward equal-frequencies ones.

In this chapter, we introduce a variant of the ctdBNs that does not suffer from this feature. Similarly to ctdBNs, it is composed of a classical BN and a graphical part specific to cope with the continuous variables. But, unlike the ctdBNs, this part is not composed of *truncated* probability density functions but rather of *untruncated* probability density functions. It is therefore called a "*conditional densities Bayesian network*" or cdBN for short. In addition to modifying the types of conditional density functions allowed, it also enables to provide several parents to each continuous variable. As we will see, this model is attractive not only for its learning potential but also for its fast inference engine as well as its robustness w.r.t. discretizations.

4.1 Definition and properties

As mentioned in the introduction, the definition of cdBNs is quite similar to that of ctdBNs, except that : i) the conditional densities may not be necessarily truncated; and ii) continuous nodes may have several parents:

Definition 4.1.1 (Bayesian networks with conditional densities (ctdBN)) *Let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ and $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$ be sets of discrete and continuous random variables respectively. Let $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ be a set of discrete variables such that to each continuous variable $\mathring{X}_i \in \mathring{\mathbf{X}}_\mathbf{C}$ corresponds a variable $X_i \in \mathbf{X_C}$. Then, a BN with conditional densities is a pair $(\mathcal{G}, \boldsymbol{\theta})$ where:*

1. $\mathcal{G} = (\mathbf{X}, \mathbf{A})$ *is a directed acyclic graph,*

2. $\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$ *and* $\mathbf{A}$ *is a set of arcs such that nodes* $\mathring{X}_i \in \mathring{\mathbf{X}}_\mathbf{C}$ *satisfy the following two properties:*

   - *they* have no children*,*
   - $\{X_i\} \subseteq \mathbf{Pa}(\mathring{X}_i) \subseteq \{X_i\} \cup \mathbf{Pa}(X_i)$*, i.e., their parents set contains necessarily* X_i*, their discrete counterpart, and it can only contain* X_i *and its parents (this condition is the key to guarantee that inference in a cdBN is about as fast as that in a classical BN),*

3. $\boldsymbol{\theta} = \boldsymbol{\theta}_\mathbf{D} \cup \boldsymbol{\theta}_\mathbf{C}$*,*

4. $\boldsymbol{\theta}_\mathbf{D} = \{P(X_i|\mathbf{Pa}(X_i))\}_{i=1}^n$ *is the set of the conditional probability tables of all the discrete variables* X_i *in* $\mathcal{G}$ *given their parents* $\mathbf{Pa}(X_i)$ *in* $\mathcal{G}$*,*

5. $\boldsymbol{\theta}_\mathbf{C} = \{f(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i))\}_{i=d+1}^n$ *is the set of the conditional densities of the continuous random variables of* $\mathring{\mathbf{X}}_\mathbf{C}$ *given their parents in the graph.*

The cdBN encodes the mixed probability distribution over $\mathbf{X}$ *as the product of all the functions in* $\boldsymbol{\theta}$*, i.e.,*

$$p(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \prod_{i=d+1}^{n} f(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i)).$$

Figure 4.1 shows an example of a cdBN: nodes in dotted circles represent continuous nodes. Note that they are only linked to their "discrete" counterpart but also to some of their parents (see Node $\mathring{X}_5$). The key point in cdBNs is that, unlike ctdBNs, the density functions are defined over the whole domain of definition of $\mathring{X}_i$. As such, they are learnt from the whole database and not just over a subset of records like the densities of ctdBNs. This is this very feature that ensures that scoring functions can be defined appropriately for cdBNs.

Proposition 4.1.1 *A cdBN is a compact representation of a mixed probability distribution.*

Proof. By definition, the cdBN encodes:

$$p(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \prod_{i=d+1}^{n} f(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i)). \tag{4.2}$$

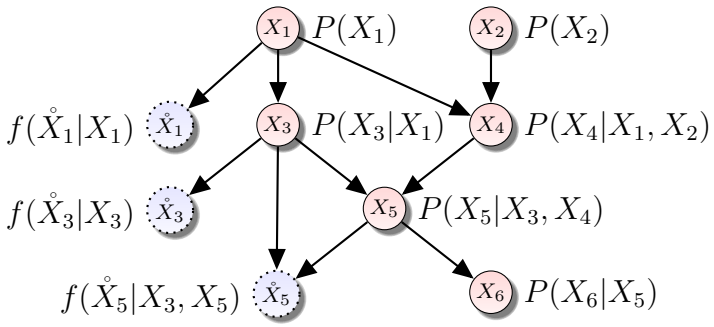


Figure 4.1: A cdBN.

As every function is non-negative, their product is also non-negative. To show that p is a mixed probability distribution, there just remains to show that:

$$\sum_{X_1} \cdots \sum_{X_n} \int_{\Omega_{\mathring{X}_{d+1}}} \cdots \int_{\Omega_{\mathring{X}_n}} p(\mathbf{X}) d\mathring{X}_n \cdots d\mathring{X}_{d+1} = 1.$$

By Eq. (4.2), this is equivalent to:

$$\begin{aligned} &\sum_{X_1} \cdots \sum_{X_n} \left(\prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \right) \times \\ &\int_{\Omega_{\mathring{X}_{d+1}}} f(\mathring{X}_{d+1}|\mathbf{Pa}(\mathring{X}_{d+1})) d\mathring{X}_{d+1} \times \cdots \times \int_{\Omega_{\mathring{X}_n}} f(\mathring{X}_n|\mathbf{Pa}(\mathring{X}_n)) d\mathring{X}_n. \end{aligned} \tag{4.3}$$

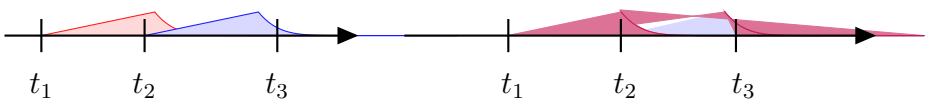
By definition, for every $i \in \{d+1, \ldots, n\}$ and every value of $\mathbf{Pa}(\mathring{X}_i)$, we have that $f(\mathring{X}_{d+1}|\mathbf{Pa}(\mathring{X}_{d+1}))$ is a probability density function, hence $\int_{\Omega_{\mathring{X}_i}} f(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i)) d\mathring{X}_i = 1$. Therefore, for every $i \in \{d+1, \ldots, n\}$, $\int_{\Omega_{\mathring{X}_i}} f(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i)) d\mathring{X}_i$ is a function of $\mathbf{Pa}(\mathring{X}_i)$ whose value is always 1. Let us denote it as $\mathbb{1}_{\mathbf{Pa}(\mathring{X}_i)}$. Then, Eq. (4.3) is equal to:

$$\sum_{X_1} \cdots \sum_{X_n} \left(\prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)) \right) \times \left(\prod_{i=d+1}^{n} \mathbb{1}_{\mathbf{Pa}(\mathring{X}_i)} \right).$$

By definition, the product inside the left parentheses is the joint probability over $\mathbf{X_D} \cup \mathbf{X_C} = \{X_1, \ldots, X_n\}$, i.e., $P(\mathbf{X_D} \cup \mathbf{X_C})$, and the product inside the right parentheses is a function defined over a subset $\mathbf{X}_E$ of $\mathbf{X_D} \cup \mathbf{X_C}$ whose value is always equal to 1. As a result $P(\mathbf{X_D} \cup \mathbf{X_C}) \times \mathbb{1}_{\mathbf{X}_E}$ is equal to probability distribution $P(\mathbf{X_D} \cup \mathbf{X_C})$. Hence, the summation over all the variables in $\mathbf{X_D} \cup \mathbf{X_C}$ is equal to 1, and the cdBN represents a mixed probability distribution. ■

4.1.1 Difference between ctdBNs and cdBNs

Fig. 4.2 illustrates the difference between the density functions used in ctdBNs and those used in cdBNs: in a ctdBN, the density function assigned to discretization interval $[t_k, t_{k+1})$ only takes into account the distribution of the values of the continuous variable defined over this interval; it never takes into account the values that are outside $[t_k, t_{k+1})$ but are still close to the interval. In a cdBN, the density function takes into account the values in interval $[t_k, t_{k+1})$ but also those lying outside (the purple parts in the figure). The latter being on the tails of the distribution, their values by the density function are lower than those in interval $[t_k, t_{k+1})$. Thus, the cdBN model can be interpreted as a robust version of ctdBNs w.r.t. discretizations. Indeed, when ctdBNs and cdBNs are exploited for decision making, the distribution of interest is never $p(\mathbf{X})$ as given in Eq. (4.1.1) but rather $p(\mathbf{X_D}, \mathring{\mathbf{X}}_\mathbf{C}) = \sum_{X_{d+1},\ldots,X_n} p(\mathbf{X})$, i.e., the discrete variables X_i corresponding to the continuous variables $\mathring{X}_i$ are unobserved (latent) variables and, as such, are marginalized out. But, after these summations, density functions become mixtures of densities. For instance, assume that $p(\mathbf{X}) = P(X_1)f(\mathring{X}_1|X_1)$ and that $\Omega_{X_1} = \{0, \ldots, g_1\}$, then, by setting $\pi_i = P(X_1 = i)$, we have that $p(\mathring{X}_1) = \sum_{i=0}^{g_1} \pi_i f(\mathring{X}_1|X_1 = i)$. As shown in Fig. 4.2.(b), these mixtures are usually quite smooth, so that small variations on the discretization cutpoints t_k have a low impact on cdBN distributions $p(\mathbf{X_D}, \mathring{\mathbf{X}}_\mathbf{C})$. Quite the opposite, in ctdBNs, small variations on the discretization cutpoints may have a much higher impact on $p(\mathbf{X_D}, \mathring{\mathbf{X}}_\mathbf{C})$ because, as shown in Fig. 4.2.(a), at the cutpoints, there exist discontinuities in the mixtures of densities.



(a) ctdBN (b) cdBN

Figure 4.2: Densities in ctdBNs v.s. densities in cdBNs.

4.1.2 Representation properties

In this section, we will show that cdBNs have at least the same expressive power as ctdBNs. Let $\mathcal{C}$ be the set of all cdBNs. First, note that, in Definition 4.1.1, the set of parents $\mathbf{Pa}(\mathring{X}_i)$ is such that $\{X_i\} \subseteq \mathbf{Pa}(\mathring{X}_i) \subseteq \{X_i\} \cup \mathbf{Pa}(X_i)$. Hence it is possible that $\mathbf{Pa}(\mathring{X}_i) = \{X_i\}$. Now, let us only consider cdBNs in which $\mathbf{Pa}(\mathring{X}_i) = \{X_i\}$. Let $\mathcal{C}_1$ be the set of such cdBNs. Clearly, $\mathcal{C}_1 \subset \mathcal{C}$. In the cdBNs

belonging to $\mathcal{C}_1$, to each continuous random variable $\mathring{X}_i$ is assigned conditional probability density function $f(\mathring{X}_i|X_i)$. Let $r_i = |\Omega_{X_i}|$ and let us consider any discretization $d_{\mathring{X}_i} : \Omega_{\mathring{X}_i} \to \{0, \ldots, r_i - 1\}$. Then, by Definition 2.1.1, there exists an increasing sequence of cutpoints $\{t_i^1, t_i^2, \ldots, t_i^{r_i-1}\} \subset \Omega_{\mathring{X}_i}$ such that:

$$d_{\mathring{X}_i}(\mathring{x}_i) = \begin{cases} 0 & \text{if } \mathring{x}_i < t_i^1, \\ k & \text{if } t_i^k \leq \mathring{x}_i < t_i^{k+1}, \text{ for all } k \in \{1, \ldots, r_i - 1\} \\ r_i - 1 & \text{if } \mathring{x}_i \geq t_i^{r_i-1} \end{cases}$$

Among all the conditional density functions $f(\mathring{X}_i|X_i)$, let us only consider those for which there exists a discretization function $d_{\mathring{X}_i} : \Omega_{\mathring{X}_i} \to \{0, \ldots, r_i - 1\}$ such that $f(\mathring{X}_i|X_i = k) = 0$ whenever $\mathring{X}_i \notin [t_i^k, t_i^{k+1})$. Then, let $\mathcal{C}_{1d}$ be the set of cdBNs of $\mathcal{C}_1$ whose conditional density functions satisfy this property for all their continuous variables. Clearly, $\mathcal{C}_{1d} \subset \mathcal{C}_1 \subset \mathcal{C}$. Now, by construction, $\mathcal{C}_{1d}$ corresponds precisely to the set of ctdBNs. As a result, whenever a mixed probability can be approximated by a ctdBN, it can also be approximated by a cdBN. Hence, propositions 3.2.2 and 3.2.3 and Corollary 3.2.1 trivially induce the following corollaries:

Corollary 4.1.1 *Let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ be a set of discrete random variables of respective domains $\{\Omega_1, \ldots, \Omega_d\}$ and let $\mathbf{\Omega_D} = \prod_{i=1}^{d} \Omega_i$ be the domain of $\mathbf{X_D}$. Let $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, ..., \mathring{X}_n\}$ be a set of continuous random variables of respective domains $\mathring{\Omega}_{d+1}, ..., \mathring{\Omega}_n$ such that none of the $\mathring{\Omega}_i$ is a singleton. Let $\mathring{\mathbf{\Omega}}_\mathbf{C} = \prod_{i=d+1}^{n} \mathring{\Omega}_i$ be the domain of $\mathring{\mathbf{X}}_\mathbf{C}$. Finally, let $\mathbf{X} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$ and $\mathbf{\Omega} = \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$.*

Let $f : \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C} \mapsto \mathbb{R}$ be a mixed probability distribution. Assume that f is Lipschitz w.r.t. the continuous variables of $\mathring{\mathbf{X}}_\mathbf{C}$, i.e., there exists a constant $M > 0$ such that, for every pair $(\mathring{x}, \mathring{y})$ of elements of $\mathring{\mathbf{\Omega}}$ such that $x_i = y_i$ for all $i \in \{1, \ldots, d\}$, $|f(\mathring{x}) - f(\mathring{y})| \leq M||\mathring{x} - \mathring{y}||$, where $||\mathring{x} - \mathring{y}||$ represents the L2-norm of vector $(\mathring{x} - \mathring{y})$.

Then, for every strictly positive real number $\epsilon < 1$, there exists a cdBN $\mathcal{B} = (\mathcal{G}, \boldsymbol{\theta})$ that approximates f up to ϵ, i.e.:

- *the nodes of $\mathcal{G}$ are $\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$, where $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ is a set of discrete variables corresponding to $\mathring{\mathbf{X}}_\mathbf{C}$; in addition, let $\mathbf{\Omega_C} = \prod_{i=d+1}^{n} \Omega_i$ and $\mathbf{\Omega} = \mathbf{\Omega_D} \times \mathbf{\Omega_C} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$ be the domains of $\mathbf{X_C}$ and $\mathbf{X}$ respectively;*

- *$\mathcal{B}$ represents a mixed probability density function $g : \mathbf{\Omega} \mapsto \mathbb{R}$ such that, for every $(y, \mathring{x}) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$, $|\sum_x g(y, x, \mathring{x}) - f(y, \mathring{x})| \leq \epsilon$, where x are the values of X, the discrete counterpart of $\mathring{X}$.*

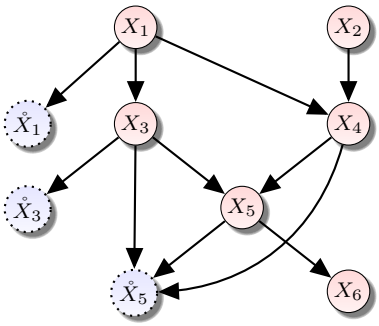
Corollary 4.1.2 *Standard distributions like, e.g., univariate and multivariate Normal distributions, Beta distributions $B(\mathring{x}, \alpha, \beta)$, with $\alpha, \beta \geq 2$, Gamma distribution $\Gamma(\mathring{x}, \alpha, \beta)$ with $\alpha > 2$, as well as their combinations by mutually independent random variables, can be approximated up to $\epsilon < 1$ by cdBNs.*

Corollary 4.1.3 *Let $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ and $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, ..., \mathring{X}_n\}$ be sets of discrete and continuous random variables respectively. Let $f : \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C} \mapsto \mathbb{R}$ be a mixed probability distribution. Assume that sets of variables $\mathbf{X_D}$ and $\mathring{\mathbf{X}}_\mathbf{C}$ can be partitioned into sets $\{\mathbf{X}_{D_1}, \ldots, \mathbf{X}_{D_k}\}$ and $\mathring{\mathbf{X}}_{C_1}, \ldots, \mathring{\mathbf{X}}_{C_k}$ respectively, and that there exist some non-negative functions $f_i : \mathbf{\Omega}_{D_i} \times \mathring{\mathbf{\Omega}}_{C_i} \mapsto \mathbb{R}$, $i = 1, \ldots, k$, such that $f(x_D, \mathring{x}_C) = \prod_{i=1}^{k} f_i(x_{D_i}, \mathring{x}_{C_i})$ for all $(x_D, \mathring{x}_C) \in \mathbf{\Omega_D} \times \mathring{\mathbf{\Omega}}_\mathbf{C}$. Then if f is Lipschitz w.r.t. the continuous variables of $\mathring{\mathbf{X}}_\mathbf{C}$, it can be approximated up to ϵ by a cdBN which has the same decomposition, i.e., sets $(\mathbf{X}_{D_i} \cup \mathbf{X}_{C_i} \cup \mathring{\mathbf{X}}_{C_i})$, $i = 1, \ldots, k$, where $\mathbf{X}_{C_i}$ are the discrete counterparts of $\mathring{\mathbf{X}}_{C_i}$, form the connected components of the cdBN's graphical structure.*

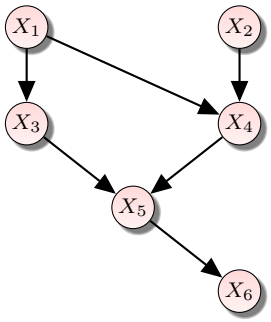
4.1.3 Inference

In Chapter 3, we have shown how to construct a junction tree for ctdBNs by appending to the junction tree of the discrete part the cliques related to the continuous nodes. In cdBNs, the principle is exactly the same. Let $\mathcal{T}$ be the junction tree of the discrete part, i.e., the one constructed from the subgraph of the cdBN defined over only the discrete nodes (including those X_i related to the continuous variables $\mathring{X}_i$). By definition of cdBNs, for any continuous node $\mathring{X}_i$, the unique clique containing $\mathring{X}_i$ is Clique $\mathcal{C}_i = \{\mathring{X}_i\} \cup \mathbf{Pa}(\mathring{X}_i)$. But as $\mathbf{Pa}(\mathring{X}_i) \subseteq \{X_i\} \cup \mathbf{Pa}(X_i)$, in order to add clique $\mathcal{C}_i$ to $\mathcal{T}$ while still ensuring that the running intersection property holds, it is sufficient to link $\mathcal{C}_i$ to a clique $\mathcal{C}_j$ containing $\{X_i\} \cup \mathbf{Pa}(X_i)$. Such a clique is guaranteed to exist because it is necessary to hold $P(X_i|\mathbf{Pa}(X_i))$. As a result, to construct the junction tree of the cdBN, it is sufficient to first create that of the discrete part and, then, to append to it the cliques of the continuous nodes, one by one, as described above. As an example, consider the cdBN of Figure 4.3a. The discrete part of the cdBN is displayed in Figure 4.3b. Therefore, the junction tree $\mathcal{T}$ of the discrete part is shown in Figure 4.4a. As mentioned above, the cliques related to the continuous nodes are cliques $\{\mathring{X}_1 X_1\}$, $\{\mathring{X}_3 X_3\}$ and $\{\mathring{X}_5 X_3 X_4 X_5\}$, and they can be added to $\mathcal{T}$ as shown in Figure 4.4b. Here, note that the result is not precisely a junction tree but a join tree because, although the running intersection property holds, it is not the case that no clique is included into another one. Actually, clique $\{X_3 X_4 X_5\}$ is included into $\{\mathring{X}_5 X_3 X_4 X_5\}$. When such a situation appears, it is easy to show that substituting the contained clique by the containing one, as shown in Figure 4.5, is sufficient to get a junction tree. However, from the inference point of view,

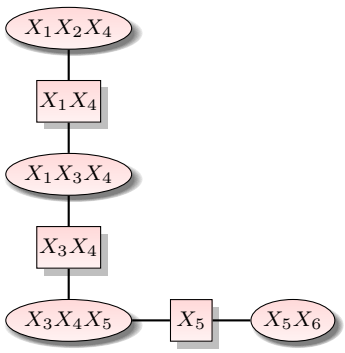
it is simpler to exploit the join tree of Figure 4.4b rather than the junction tree of Figure 4.5 because, in the former, the same algorithm as the one provided for ctdBNs can be used. When using the junction tree instead, during the computations of every message, it becomes necessary to distinguish the functions defined over continuous variables from those defined only over discrete variables (because the processes to eliminate variables differ). This is therefore somewhat more complicated to define the inference algorithm in the junction tree rather than to use just the algorithm defined over ctdBNs.



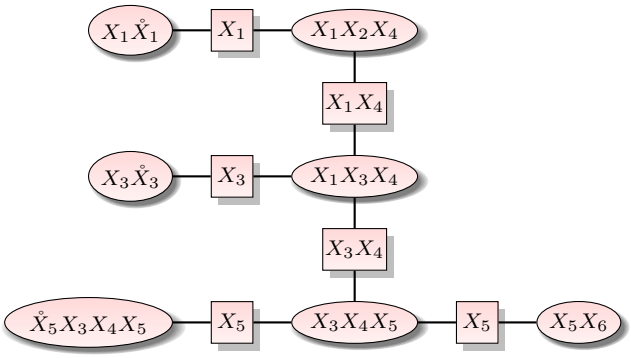
(a) A simple cdBN.



(b) The discrete part of the cdBN.



(a) Junction tree $\mathcal{T}$.



(b) The join tree of the whole cdBN.

As mentioned above, in the join tree, the inference algorithm is precisely the same as the one provided for ctdBNs. The complexity of inference can be derived in a similar way as in ctdBNs. However, in the worst case, to each continuous variable $\mathring{X}_i$ is assigned a set of parents $\mathbf{Pa}(\mathring{X}_i) = \{X_i\} \cup \mathbf{Pa}(X_i)$, which may induce that all the cliques assigned to the continuous variables include their neighboring cliques in the join tree. This, in turn, directly implies that the inference complexity w.r.t. the whole cdBN is defined as follows:

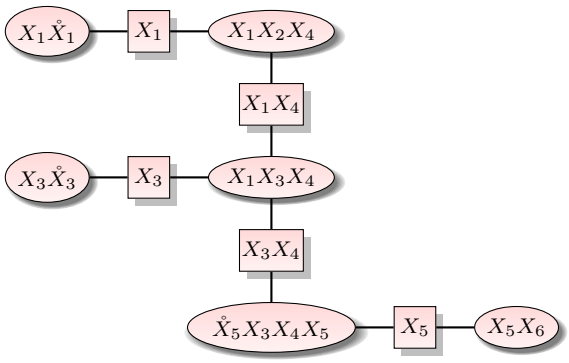


Figure 4.5: The junction tree of the whole cdBN.

Proposition 4.1.2 *Let w be the treewidth of the junction tree $\mathcal{T}$ defined over the discrete part of the cdBN. Let k denote the maximum domain size of the discrete random variables. Finally, let n be the number of random variables in the cdBN and let $\overline{I}$ be the average complexity of computing one integral, as in Equation (3.6), and $\overline{J}$ the average complexity of computing the product as in Equation (3.7). Then the complexity of computing all the marginal posterior distributions of all the random variables is in $O(nk^{w+1}(1+\overline{I}+\overline{J}))$.*

4.2 Learning CdBNs

The main advantage of cdBNs over ctdBNs lies in their structure learning from data. As we have seen in a preceding subsection, the truncated densities used by ctdBNs do not allow for an easy adaptation of the classical scores used in BN structure learning like K2, BDeu, BIC, *etc.* [Cooper and Herskovits, 1992a, Geiger and Heckerman, 1995, Heckerman et al., 1995, Heckerman, 1995, Schwarz, 1978]. Fortunately, with cdBNs, these scores can be adapted as well as the learning algorithm. To understand it, consider the cdBN of Figure 4.3a learnt from a database $\mathring{\mathcal{D}}$. In this database, only variables $\mathring{X}_1, X_2, \mathring{X}_3, X_4, \mathring{X}_5, X_6$ are observed, variables X_1, X_3, X_5 are latent, hidden. So, learning cdBNs boils down to learning a model with hidden variables, which is precisely what Structural EM (SEM) is made for [Friedman, 1998, Peña et al., 2000]. This is summarized in Algorithm 4.1.

Some details must be given about this algorithm. For this purpose, assume that Database $\mathring{\mathcal{D}}$ contains N records, each one being defined over discrete random variables $\mathbf{X_D} = \{X_1, \ldots, X_d\}$ and continuous variables $\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$. We assume that the database is complete, i.e., for each record, the values of all the random variables are observed. Finally, we let $\mathbf{X_C} = \{X_{d+1}, \ldots, X_n\}$ and

Algorithm 4.1: Structural EM for learning cdBNs.

Input: Database $\mathring{\mathcal{D}}$
Output: a ctdBN $\mathcal{B} = (\mathcal{G}, \mathbf{\Theta})$
1 $t \leftarrow 0$
2 $\mathcal{G}_0 \leftarrow$ an initial cdBN graphical structure
3 $\mathbf{\Theta}_0 \leftarrow$ MAP parameters of $\mathcal{G}_0$ given $\mathring{\mathcal{D}}$
4 **repeat**
// **E step**
5 **foreach** $\mathcal{G}$ *in the neighborhood* $\mathbf{N}(\mathcal{G}_t)$ *of* $\mathcal{G}_t$ **do**
6 Assign score $Sc(\mathcal{G}) = \log p(\mathcal{G}|\mathring{\mathcal{D}})$ to $\mathcal{G}$
// **M step**
7 $\mathcal{G}_{t+1} \leftarrow \text{Argmax}_{\mathcal{G} \in \mathbf{N}(\mathcal{G}_t)} Sc(\mathcal{G})$
8 $\mathbf{\Theta}_{t+1} \leftarrow$ MAP parameters of $\mathcal{G}_{t+1}$ given $\mathring{\mathcal{D}}$
9 $t \leftarrow t + 1$
10 **until** $\mathcal{G}_{t+1} = \mathcal{G}_t$;
11 **return** *cdBN* $\mathcal{B} = (\mathcal{G}_t, \mathbf{\Theta}_t)$

$\mathbf{X} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_\mathbf{C}$ be the set of latent variables and the set of all the variables of our model respectively. On Line 2, an initial graph is provided. Usually, in BNs, this is an empty graph, i.e., a graph without any arc. Here, the equivalent for cdBNs is a graph $\mathcal{G}_0 = (\mathbf{X}, \mathbf{A}_0)$, where $\mathbf{A}_0 = \{(X_i, \mathring{X}_i) : i = d+1, \ldots, n\}$, i.e., the graph contains only arcs from the latent variables to their continuous counterparts. On Line 3, the optimal parameters of the conditional densities as well as the marginal probabilities of the the X_i's need be determined. All the X_i's, $i \leq d$ are independent from the other variables, so the determination of their distribution $P(X_i)$ by MAP is a classic, notably if the *a priori* over the parameters of their CPT is a Dirichlet distribution [Heckerman et al., 1995, Heckerman, 1995]. All the pairs $(X_i, \mathring{X}_i)$, $i = d+1, \ldots, n$ are mutually independent, so the parameters of their distributions can be determined independently. For variables $\mathring{X}_i$, we first need to assume that the conditional densities belong to a given family of distributions. In the following, we assume that this family is that of the Normal distributions $\mathcal{N}(\mu, \tau^{-1})$, where $\tau = 1/\sigma^2$ is called the *precision*. It is well-known that the conjugate prior of such distributions is the Normal-Gamma function $N\Gamma(\mu_0, \lambda_0, \alpha_0, \beta_0)$, which can be used as the prior over the parameters of the conditional densities. Now, note that the distribution of every continuous variable $\mathring{X}_i$ is equal to $\sum_{X_i} P(X_i) f(\mathring{X}_i|X_i)$, which is a mixture of Normal distributions. So the optimal parameters for f are determined by Maximum Likelihood Estimation (MLE) as those of a mixture of Gaussians and, by MAP, as a mixture

of Normal-Gamma functions. This can be performed efficiently by a classical EM algorithm.

In Line 6, $Sc(\mathcal{G}) = \log p(\mathcal{G}|\mathring{\mathcal{D}})$ should be computed. By Bayes rule, we have that $Sc(\mathcal{G}) = \log p(\mathring{\mathcal{D}}|\mathcal{G}) + \log(p(\mathcal{G})/p(\mathcal{D}))$. Assuming a uniform prior over all the graphical structures, the second term is a constant and needs not be taken into account. Unfortunately, $\log p(\mathring{\mathcal{D}}|\mathcal{G})$ cannot be computed in closed form because some variables are not observed, so, in the SEM algorithm, it is approximated as follows: let $\mathbf{X}_{\mathbf{D}}^{(m)}, \mathring{\mathbf{X}}_{\mathbf{C}}^{(m)}$ (resp. $\mathbf{x}_{\mathbf{C}}^{(m)}$) be the observed (resp. unobserved) variables in the mth record of the database, and let $\mathbf{X}_{\mathbf{C}}^{(\mathcal{D})} = \{\mathbf{X}_{\mathbf{C}}^{(m)}\}_{m=1}^{N}$, $\mathbf{X}_{\mathbf{D}}^{(\mathcal{D})} = \{\mathbf{X}_{\mathbf{D}}^{(m)}\}_{m=1}^{N}$ and $\mathring{X}_{\mathbf{C}}^{(\mathcal{D})} = \{\mathring{X}_{\mathbf{C}}^{(m)}\}_{m=1}^{N}$, be the sets over *all* the records. Then, assuming the cdBN determined at step t of the learning algorithm is $\mathcal{B}_t = (\mathcal{G}_t, \mathbf{\Theta}_t)$, which represents distribution $p_{\mathcal{B}_t}(\cdot)$, the Structural EM score assigned to $\mathcal{G}$ is equal to:

$$
\begin{aligned}
Sc(\mathcal{G}) &\approx \sum_{\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}} p_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathbf{x}_{\mathbf{D}}^{(\mathcal{D})}, \mathring{x}_{\mathbf{C}}^{(\mathcal{D})}, \mathcal{G}_t, \mathbf{\Theta}_t) \times \log p(\mathbf{x}_{\mathbf{D}}^{(\mathcal{D})}, \mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}, \mathring{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathcal{G}). \\
&\approx \sum_{\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}} p_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathbf{x}_{\mathbf{D}}^{(\mathcal{D})}, \mathring{x}_{\mathbf{C}}^{(\mathcal{D})}, \mathcal{G}_t, \mathbf{\Theta}_t) \times \\
&\qquad \left[\sum_{i=1}^{n} Sc(X_i|\mathbf{Pa}(X_i)) + \sum_{i=d+1}^{n} Sc(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i)) \right]
\end{aligned}
\tag{4.4}
$$

where the last terms correspond to the scores of every node of the cdBN given their parents. The part corresponding to $Sc(X_i|\mathbf{Pa}(X_i))$, $i = 1, \ldots, n$, concerns only discrete variables and can therefore be computed exactly as in the classical SEM algorithm [Friedman, 1998]. We will describe how to compute the score $Sc(\mathring{X}_i|\mathbf{Pa}(\mathring{X}_i))$ related to the continuous part in the next subsection.

On Lines 5 and 7, we need to define the neighborhood of a graph $\mathcal{G}$. If we just consider the discrete part $\mathcal{G}_D$ of $\mathcal{G}$, the neighborhood can be classically defined as all the graphs that can result from the addition of an arc to $\mathcal{G}_D$, from the removal of an arc from $\mathcal{G}_D$ or from an arc reversal in $\mathcal{G}_D$. In a similar fashion, for the continuous part, i.e., the set of parents of the $\mathring{X}_i$'s, the neighborhood consists of adding a new parent, while satisfying constraint $\{X_i\} \subseteq \mathbf{Pa}(\mathring{X}_i) \subseteq \{X_i\} \cup \mathbf{Pa}(X_i)$, or removing an already existing parent (except X_i). Overall, the neighborhood of graph $\mathcal{G}$ is the union of the discrete and continuous neighborhoods. To speed-up searching, one may restrict this neighborhood, considering that constraining $\mathbf{Pa}(\mathring{X}_i)$ to be equal to precisely to $\{X_i\} \cup \mathbf{Pa}(X_i)$ results in the cdBN best-fitting to data $\mathring{\mathcal{D}}$.

4.2.1 A new score for cdBNs

Clearly, for using search-based structure learning approaches, we need to define a score well-suited for cdBNs, i.e., this score needs to take into account both the discrete and the continuous parts of the cdBN. The goal of this subsection is to propose such a score. For this purpose, we need to introduce a few additional notations. In the sequel, let $\mathring{\mathbf{X}}_{\mathbf{O}} = \mathbf{X_D} \cup \mathring{\mathbf{X}}_{\mathbf{C}}$ denote the set of observed random variables in dataset $\mathcal{D}$ and let $\mathring{\mathbf{X}} = \mathbf{X_D} \cup \mathbf{X_C} \cup \mathring{\mathbf{X}}_{\mathbf{C}}$ be the set of all the variables in the cdBN. Next, we will consider that each record of database $\mathcal{D}$ is an instantiation of some random variables whose distribution is the same as that of $\mathring{\mathbf{X}}$. Thus, we denote by superscript (m) (resp. $(\mathcal{D})$) the variables that refer to the mth record of the database (resp. all the records of the database). For instance, $\mathbf{X}_{\mathbf{C}}^{(\mathcal{D})} = \{\mathbf{X}_{\mathbf{C}}^{(m)}\}_{m=1}^{N}$ and $\mathbf{X}_{\mathbf{C}}^{(m)}$ correspond to the set of unobserved variables in all the records of $\mathcal{D}$ and in the mth record of $\mathcal{D}$ respectively. Let $\mathbf{X}_{\mathbf{FC}_i} = (\{X_i\} \cup \mathbf{Pa}_{\mathcal{G}}(X_i)) \cap \mathbf{X_C}$, $i = 1, \ldots, n$, and $\mathring{\mathbf{X}}_{\mathbf{PC}_i} = \{\mathring{X}_i\} \cup (\mathbf{Pa}_{\mathcal{G}}(\mathring{X}_i) \cap \mathbf{X_C})$, $i = d+1, \ldots, n$, represent the set of unobserved variables among X_i or $\mathring{X}_i$ and its parents in graph $\mathcal{G}$. Finally, for any $i \in \{1, \ldots, n\}$, let $\mathbf{X}_{\mathbf{PC}_i} = \mathbf{X}_{\mathbf{FC}_i} \backslash \{X_i\}$ be the set of the unobserved parents of X_i, let $\mathbf{X}_{\overline{\mathbf{PC}}_i} = \mathbf{Pa}_{\mathcal{G}}(X_i) \setminus \mathbf{X}_{\mathbf{PC}_i}$ be the set of observed parents and let $\mathbf{X}_{\overline{\mathbf{OC}}_i} = \mathring{\mathbf{X}}_{\mathbf{O}} \backslash (\{X_i\} \cup \mathbf{X}_{\overline{\mathbf{PC}}_i})$ be all the observed variables but X_i and its parents. We can now provide our new score:

Proposition 4.2.1 *Assume that the cdBN determined at step t of the learning algorithm is $\mathcal{B}_t = (\mathcal{G}_t, \mathbf{\Theta}_t)$ and that it represents distribution $P_{\mathcal{B}_t}(\cdot)$. Let $\mathcal{G}$ be the structure of a cdBN. Let us assume parameter independence. In addition, let the prior over the parameters of the CPTs assigned to the discrete and latent variables be a Dirichlet distribution of hyperparameters $(\alpha_{ij_1j_2 1}, \ldots, \alpha_{ij_1j_2 r_i})$. Finally, assume that the conditional densities of the continuous variables of the cdBN are Normal distributions and that the prior over their parameters is a Normal-Gamma distribution $N\Gamma(\rho_{ij_1j_2}, \lambda_{ij_1j_2}, \alpha_{ij_1j_2}, \beta_{ij_1j_2})$.*

Then the score assigned to $\mathcal{G}$ is equal to $S(\mathcal{G}) = S_1(\mathcal{G}) + S_2(\mathcal{G})$, where:

$$S_1(\mathcal{G}) = \sum_{i=1}^{n}\sum_{j_1=1}^{q_{i1}}\sum_{j_2=1}^{q_{i2}}\left[\log\left(\frac{\Gamma(\alpha_{ij_1j_2})}{\Gamma(N_{ij_1j_2}+\alpha_{ij_1j_2})}\right)+\sum_{k=1}^{r_i}\log\left(\frac{\Gamma(N_{ij_1j_2k}+\alpha_{ij_1j_2k})}{\Gamma(\alpha_{ij_1j_2k})}\right)\right]$$

$$S_2(\mathcal{G}) = \sum_{i=1}^{n}\sum_{j_1=1}^{q_{i1}}\sum_{j_2=1}^{q_{i2}}\log\left[\frac{1}{\sqrt{2\pi}^{\mathring{N}_{ij_1j_2}}}\frac{\beta_{ij_1j_2}^{\alpha_{ij_1j_2}}}{\gamma_{ij_1j_2}^{\alpha_{ij_1j_2}+\frac{\mathring{N}_{ij_1j_2}}{2}}}\frac{\Gamma(\alpha_{ij_1j_2}+\frac{\mathring{N}_{ij_1j_2}}{2})}{\Gamma(\alpha_{ij_1j_2})}\frac{\sqrt{\lambda_{ij_1j_2}}}{\sqrt{\lambda_{ij_1j_2}+\mathring{N}_{ij_1j_2}}}.\right]$$

$$N_{ij_1j_2k} = \begin{cases}\displaystyle\sum_{m:\mathbf{x}_{\mathbf{PC}_i}^{(m)}=j_2,x_i^{(m)}=k} P_{\mathcal{B}_t}(\mathbf{X}_{\mathbf{PC}_i}=j_1|\mathring{\mathbf{x}}_{\mathbf{OC}_i}^{(m)},\mathbf{x}_{\mathbf{PC}_i}^{(m)},x_i^{(m)}) \text{ if } i\le d\\ \displaystyle\sum_{m:\mathbf{x}_{\mathbf{PC}_i}^{(m)}=j_2} P_{\mathcal{B}_t}(\mathbf{X}_{\mathbf{PC}_i}=j_1,x_i^{(m)}|\mathring{\mathbf{x}}_{\mathbf{OC}_i}^{(m)},\mathbf{x}_{\mathbf{PC}_i}^{(m)}) \text{ if } i>d\end{cases}$$

$$\mathring{N}_{ij_1j_2k} = \sum_{m:\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(m)}=j_2,\mathring{x}_i^{(m)}=k} P_{\mathcal{B}_t}(\mathring{\mathbf{x}}_{\mathbf{PC}_i}=j_1|\mathring{\mathbf{x}}_{\mathbf{OC}_i}^{(m)},\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(m)},\mathring{x}_i^{(m)})$$

Proof. The score of structure $\mathcal{G}$ is equal to $\log P(\mathring{\mathcal{D}}|\mathcal{G})$. Assuming the cdBN determined at step t of the learning algorithm is $\mathcal{B}_t = (\mathcal{G}_t, \mathbf{\Theta}_t)$, which represents distribution $P_{\mathcal{B}_t}(\cdot)$, SEM approximates this score as:

$$S(\mathcal{G}) = \sum_{\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}} P_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathring{\mathbf{x}}_{\mathbf{O}}^{(\mathcal{D})})\log P(\mathring{\mathbf{x}}^{(\mathcal{D})}|\mathcal{G}),$$

where $\mathring{\mathbf{X}}_{\mathbf{O}} = \mathbf{X}_{\mathbf{D}} \cup \mathring{\mathbf{X}}_{\mathbf{C}}$ is the set of observed random variables in dataset $\mathcal{D}$. This can be rewritten as:

$$S(\mathcal{G}) = \log\prod_{\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}}\left[\int_{\mathbf{\Theta}} P(\mathring{\mathbf{x}}^{(\mathcal{D})}|\mathcal{G},\mathbf{\Theta})\pi(\mathbf{\Theta}|\mathcal{G})d\mathbf{\Theta}\right]^{P_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathring{\mathbf{x}}_{\mathbf{O}}^{(\mathcal{D})})}$$

where $\mathbf{\Theta}$ corresponds to the potential parameters of a cdBN of structure $\mathcal{G}$ and $\pi(\cdot|\mathcal{G})$ is their prior. Let $\boldsymbol{\theta}_i$, $i = 1,\ldots,n$, denote the set of parameters assigned to CPT $P(X_i|\mathbf{Pa}_{\mathcal{G}}(X_i))$ and let $\boldsymbol{\nu}_i$, $i = d+1,\ldots,n$, denote the set of parameters assigned to $f(\mathring{X}_i|\mathbf{Pa}_{\mathcal{G}}(\mathring{X}_i))$. Assuming parameter independence, the score can be rewritten as:

$$\log\prod_{\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}}\left[\prod_{i=1}^{n}\int_{\boldsymbol{\theta}_i}P(x_i^{(\mathcal{D})}|\mathbf{Pa}_{\mathcal{G}}(x_i)^{(\mathcal{D})},\boldsymbol{\theta}_i)\pi(\boldsymbol{\theta}_i|\mathcal{G})d\boldsymbol{\theta}_i\times\right.$$
$$\left.\prod_{i=d+1}^{n}\int_{\boldsymbol{\nu}_i}f(\mathring{x}_i^{(\mathcal{D})}|\mathbf{Pa}_{\mathcal{G}}(\mathring{x}_i)^{(\mathcal{D})},\boldsymbol{\nu}_i)\pi(\boldsymbol{\nu}_i|\mathcal{G})d\boldsymbol{\nu}_i\right]^{P_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{C}}^{(\mathcal{D})}|\mathring{\mathbf{x}}_{\mathbf{O}}^{(\mathcal{D})})}$$

By marginalizing out, for each integral, all the variables that do not belong to it, the score is equal to:

$$\sum_{i=1}^{n} \log \prod_{\mathbf{x}_{\mathbf{FC}_i}^{(\mathcal{D})}} \left[\int_{\boldsymbol{\theta}_i} P(x_i^{(\mathcal{D})} | \mathbf{Pa}_{\mathcal{G}}(x_i)^{(\mathcal{D})}, \boldsymbol{\theta}_i) \, \pi(\boldsymbol{\theta}_i | \mathcal{G}) d\boldsymbol{\theta}_i \right]^{P_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{FC}_i}^{(\mathcal{D})} | \mathring{\mathbf{x}}_{\mathbf{O}}^{(\mathcal{D})})} +$$
$$\sum_{i=d+1}^{n} \log \prod_{\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(\mathcal{D})}} \left[\int_{\boldsymbol{\nu}_i} f(\mathring{x}_i^{(\mathcal{D})} | \mathbf{Pa}_{\mathcal{G}}(\mathring{x}_i)^{(\mathcal{D})}, \boldsymbol{\nu}_i) \, \pi(\boldsymbol{\nu}_i | \mathcal{G}) d\boldsymbol{\nu}_i \right]^{P_{\mathcal{B}_t}(\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(\mathcal{D})} | \mathring{\mathbf{x}}_{\mathbf{O}}^{(\mathcal{D})})}$$

where $\mathbf{X}_{\mathbf{FC}_i} = (\{X_i\} \cup \mathbf{Pa}_{\mathcal{G}}(X_i)) \cap \mathbf{X}_{\mathbf{C}}$, $i = 1, \ldots, n$, and $\mathring{\mathbf{X}}_{\mathbf{PC}_i} = \{\mathring{X}_i\} \cup (\mathbf{Pa}_{\mathcal{G}}(\mathring{X}_i) \cap \mathbf{X}_{\mathbf{C}})$, $i = d+1, \ldots, n$, represent the set of unobserved variables among X_i or $\mathring{X}_i$ and its parents in graph $\mathcal{G}$. Eq. (8) of [Friedman, 1998] suggests to approximate the product over $\mathbf{x}_{\mathbf{FC}_i}^{(\mathcal{D})}$ and $\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(\mathcal{D})}$ of the integrals by the integrals of these products, which leads to:

$$\sum_{i=1}^{n} \log \int_{\boldsymbol{\theta}_i} \prod_{m=1}^{N} \prod_{\mathbf{x}_{\mathbf{FC}_i}^{(m)}} P(x_i^{(m)} | \mathbf{Pa}_{\mathcal{G}}(x_i)^{(m)}, \boldsymbol{\theta}_i)^{P_{\mathcal{B}_t}(\mathbf{x}_{\mathbf{FC}_i}^{(m)} | \mathring{\mathbf{x}}_{\mathbf{O}}^{(m)})} \times \pi(\boldsymbol{\theta}_i | \mathcal{G}) d\boldsymbol{\theta}_i +$$
$$\sum_{i=d+1}^{n} \log \int_{\boldsymbol{\nu}_i} \prod_{m=1}^{N} \prod_{\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(m)}} f(\mathring{x}_i^{(m)} | \mathbf{Pa}_{\mathcal{G}}(\mathring{x}_i)^{(m)}, \boldsymbol{\nu}_i)^{P_{\mathcal{B}_t}(\mathring{\mathbf{x}}_{\mathbf{PC}_i}^{(m)} | \mathring{\mathbf{x}}_{\mathbf{O}}^{(m)})} \times \pi(\boldsymbol{\nu}_i | \mathcal{G}) d\boldsymbol{\nu}_i$$

Hence, $S(\mathcal{G}) = S_1 + S_2$, where S_1 corresponds to the first sum of logs above and S_2 to the second one. For any $i \in \{1, \ldots, n\}$, let $\mathbf{X}_{\mathbf{PC}_i} = \mathbf{X}_{\mathbf{FC}_i} \backslash \{X_i\}$ be the set of the unobserved parents of X_i, let $\mathbf{X}_{\overline{\mathbf{PC}}_i} = \mathbf{Pa}_{\mathcal{G}}(X_i) \setminus \mathbf{X}_{\mathbf{PC}_i}$ be the set of observed parents and let $\mathbf{X}_{\overline{\mathbf{OC}}_i} = \mathring{\mathbf{X}}_{\mathbf{O}} \backslash (\{X_i\} \cup \mathbf{X}_{\overline{\mathbf{PC}}_i})$ be all the observed variables but X_i and its parents. Denote by r_i, q_{i1} and q_{i2} the respective domain sizes of X_i, $\mathbf{X}_{\mathbf{PC}_i}$ and $\mathbf{X}_{\overline{\mathbf{PC}}_i}$, and by $\theta_{ij_1j_2k}$ the value of Parameter $P(X_i = k | \mathbf{X}_{\mathbf{PC}_i} = j_1, \mathbf{X}_{\overline{\mathbf{PC}}_i} = j_2)$ and let:

$$N_{ij_1j_2k} = \begin{cases} \displaystyle\sum_{m : \mathbf{x}_{\overline{\mathbf{PC}}_i}^{(m)} = j_2, x_i^{(m)} = k} P_{\mathcal{B}_t}(\mathbf{X}_{\mathbf{PC}_i} = j_1 | \mathring{\mathbf{x}}_{\overline{\mathbf{OC}}_i}^{(m)}, \mathbf{x}_{\overline{\mathbf{PC}}_i}^{(m)}, x_i^{(m)}) \text{ if } i \leq d \\ \displaystyle\sum_{m : \mathbf{x}_{\overline{\mathbf{PC}}_i}^{(m)} = j_2} P_{\mathcal{B}_t}(\mathbf{X}_{\mathbf{PC}_i} = j_1, x_i^{(m)} | \mathring{\mathbf{x}}_{\overline{\mathbf{OC}}_i}^{(m)}, \mathbf{x}_{\overline{\mathbf{PC}}_i}^{(m)}) \qquad \text{if } i > d \end{cases}$$

with, by abuse, $N_{ij_1j_2k}$ is equal to the number of records in the database in which $\mathbf{x}_{\overline{\mathbf{PC}}_i} = j_2$ and $x_i = k$ when $\mathbf{X}_{\mathbf{PC}_i} = \emptyset$ and $i \leq d$. Finally, for every i, j_1, j_2,

assume that the prior over $(\theta_{ij_1j_21}, \ldots, \theta_{ij_1j_2r_i})$ is a Dirichlet distribution of hyper-parameters $(\alpha_{ij_1j_21}, \ldots, \alpha_{ij_1j_2r_i})$. Then, by local parameter independence, Score S_1 above is equal to:

$$\begin{aligned} S_1 &= \sum_{i=1}^{n}\sum_{j_1=1}^{q_{i1}}\sum_{j_2=1}^{q_{i2}} \log \int_{\boldsymbol{\theta}_{ij_1j_2}} \prod_{k=1}^{r_i} \theta_{ij_1j_2k}^{N_{ij_1j_2k}+\alpha_{ij_1j_2k}} d\boldsymbol{\theta}_{ij_1j_2} + c \\ &= \sum_{i=1}^{n}\sum_{j_1=1}^{q_{i1}}\sum_{j_2=1}^{q_{i2}} \left[\log\left(\frac{\Gamma(\alpha_{ij_1j_2})}{\Gamma(N_{ij_1j_2}+\alpha_{ij_1j_2})}\right) + \sum_{k=1}^{r_i} \log\left(\frac{\Gamma(N_{ij_1j_2k}+\alpha_{ij_1j_2k})}{\Gamma(\alpha_{ij_1j_2k})}\right)\right], \end{aligned}$$

where c is the normalization constant of the Dirichlet distribution, $\boldsymbol{\theta}_{ij} = \{\theta_{ij_1j_2}\}_{k=1}^{r_i}$, $N_{ij_1j_2} = \sum_{k=1}^{r_i} N_{ij_1j_2k}$ and $\alpha_{ij_1j_2} = \sum_{k=1}^{r_i} \alpha_{ij_1j_2k}$. Let us now rewrite Score S_2. As above, for every $i = d+1, \ldots, n$, let $\mathring{\mathbf{X}}_{\mathbf{PC}_i} = \mathbf{Pa}_{\mathcal{G}}(\mathring{X}_i) \cap \mathbf{X_C}$, let $\mathring{\mathbf{X}}_{\overline{\mathbf{PC}}_i} = \mathbf{Pa}_{\mathcal{G}}(\mathring{X}_i) \setminus \mathring{\mathbf{X}}_{\mathbf{PC}_i}$ and $\mathring{\mathbf{X}}_{\overline{\mathbf{OC}}_i} = \mathring{\mathbf{X}}_{\mathbf{O}} \backslash (\{\mathring{X}_i\} \cup \mathring{\mathbf{X}}_{\overline{\mathbf{PC}}_i})$. Here, note that $\mathring{\mathbf{X}}_{\mathbf{PC}_i}$, $\mathring{\mathbf{X}}_{\overline{\mathbf{PC}}_i}$ and $\mathring{\mathbf{X}}_{\overline{\mathbf{OC}}_i}$ are sets of discrete variables. In addition, let r_i be the number of different values observed for $\mathring{X}_i$, and let q_{i1} and q_{i2} be defined as above. Then S_2 is equal to:

$$\begin{aligned} S_2 = \sum_{i=1}^{n}\sum_{j_1=1}^{q_{i_1}}\sum_{j_2=1}^{q_{i_2}} \log \int_{\boldsymbol{\nu}_{ij_1j_2}} \prod_{k=1}^{r_i} \pi(\boldsymbol{\nu}_{ij_1j_2}|\mathcal{G}) \times \\ f(\mathring{x}_i = k|\mathring{\mathbf{X}}_{\mathbf{PC}_i} = j_1, \mathring{\mathbf{X}}_{\overline{\mathbf{PC}}_i} = j_2, \boldsymbol{\nu}_{ij_1j_2})^{\mathring{N}_{ij_1j_2k}} d\boldsymbol{\nu}_{ij_1j_2} \end{aligned} \tag{4.5}$$

where $\boldsymbol{\nu}_{ij_1j_2} = \{\boldsymbol{\nu}_{ij_1j_2k}\}_{k=1}^{r_i}$ and

$$\mathring{N}_{ij_1j_2k} = \sum_{m:\mathring{\mathbf{x}}^{(m)}_{\overline{\mathbf{PC}}_i}=j_2, \mathring{x}_i^{(m)}=k} P_{\mathcal{B}_t}(\mathring{\mathbf{x}}_{\mathbf{PC}_i} = j_1 | \mathring{\mathbf{x}}^{(m)}_{\overline{\mathbf{OC}}_i}, \mathring{\mathbf{x}}^{(m)}_{\overline{\mathbf{PC}}_i}, \mathring{x}_i^{(m)}).$$

Now, for every j_1, j_2, assume that $f(\mathring{x}_i|\mathring{\mathbf{X}}_{\mathbf{PC}_i} = j_1, \mathring{\mathbf{X}}_{\overline{\mathbf{PC}}_i} = j_2)$ is the density of Normal distribution $\mathcal{N}(\mu_{ij_1j_2}, \tau^{-1}_{ij_1j_2})$, where $\tau_{ij_1j_2}$ is the precision. So $\boldsymbol{\nu}_{ij_1j_2} = (\mu_{ij_1j_2}, \tau_{ij_1j_2})$. In addition, assume that the prior over $\boldsymbol{\nu}_{ij_1j_2}$ is a Normal-Gamma function $N\Gamma(\rho_{ij_1j_2}, \lambda_{ij_1j_2}, \alpha_{ij_1j_2}, \beta_{ij_1j_2})$. Let us rewrite the log expression in Eq. (4.5) for a given value of i, j_1, j_2. For clarity reasons, we will subsequently drop subscript "ij_1j_2" in all parameters. Hence, the log is equal to:

$$\begin{aligned} \log \int_{\boldsymbol{\nu}} \prod_{k=1}^{r_i} \left[\frac{\sqrt{\tau}}{\sqrt{2\pi}} \exp\left(-\tfrac{\tau}{2}(x_i^k - \mu)^2\right)\right]^{\mathring{N}_k} \frac{\beta^\alpha \sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \\ \tau^{\alpha-1/2} \exp(-\beta\tau) \exp\left(-\tfrac{1}{2}\lambda\tau(\mu-\rho)^2\right) d\boldsymbol{\nu}, \end{aligned} \tag{4.6}$$

where x_i^k stands for the kth value observed for Variable X_i in the database and $\mathring{N}_k$ is a shortcut for $\mathring{N}_{ij_1j_2k}$. Let $\mathring{N} = \sum_{k=1}^{r_i} \mathring{N}_k$ and let $\overline{x}_i = \frac{1}{\mathring{N}} \sum_{k=1}^{r_i} \mathring{N}_k x_i^k$ and

$s = \frac{1}{\mathring{N}} \sum_{k=1}^{r_i} \mathring{N}_k (x_i^k - \overline{x}_i)^2$. Then Eq. (4.6) is equal to:

$$\log \int_\tau \int_\mu \left[\frac{\sqrt{\tau}}{\sqrt{2\pi}}\right]^{\mathring{N}} \exp\left(-\tfrac{\tau}{2}(\mathring{N}s + \mathring{N}(\overline{x}_i - \mu)^2)\right) \frac{\beta^\alpha \sqrt{\lambda}}{\Gamma(\alpha)\sqrt{2\pi}} \tau^{\alpha - 1/2} \exp(-\beta\tau) \exp\left(-\tfrac{1}{2}\lambda\tau(\mu - \rho)^2\right) d\mu \, d\tau.$$

which is equal to:

$$\log \left[\frac{\beta^\alpha \sqrt{\lambda}}{\Gamma(\alpha)(\sqrt{2\pi})^{\mathring{N}+1}}\right] + \log \int_\tau \exp\left(-\tau \tfrac{\mathring{N}s}{2} + \beta\right) \tau^{\alpha + \frac{\mathring{N}-1}{2}} \int_\mu \exp\left(-\frac{\tau}{2}[\mathring{N}(\overline{x}_i - \mu)^2 + \lambda(\mu - \rho)^2]\right) d\mu \, d\tau.$$

The μ-integral can be rewritten as:

$$\begin{aligned} &\int_\mu \exp\left[-\tfrac{\tau}{2}\left((\lambda + \mathring{N})\Big(\mu - \tfrac{\lambda\rho + \mathring{N}\overline{x}_i}{\lambda + \mathring{N}}\Big)^2 + \tfrac{\lambda \mathring{N}(\overline{x}_i - \rho)^2}{\lambda + \mathring{N}}\right)\right] d\mu \\ &= \exp\left[-\tfrac{\tau}{2}\tfrac{\lambda \mathring{N}(\overline{x}_i - \rho)^2}{\lambda + \mathring{N}}\right] \sqrt{2\pi}\overline{\sigma} \int_\mu \tfrac{1}{\sqrt{2\pi}\overline{\sigma}} \exp\left[-\tfrac{1}{2}\Big(\tfrac{\mu - \overline{\mu}}{\overline{\sigma}}\Big)^2\right] d\mu \end{aligned}$$

with $\overline{\mu} = \frac{\lambda\rho + \mathring{N}\overline{x}_i}{\lambda + \mathring{N}}$ and $\overline{\sigma} = 1/\sqrt{\tau(\lambda + \mathring{N})}$. The last integral is that of a normal distribution, hence it is equal to 1. Hence Eq. (4.6) is equal to:

$$\log \left[\tfrac{\beta^\alpha \sqrt{\lambda}}{\Gamma(\alpha)(\sqrt{2\pi})^{\mathring{N}+1}}\right] + \log \tfrac{\sqrt{2\pi}}{\sqrt{\lambda + \mathring{N}}} \int_\tau \tau^{\alpha + \frac{\mathring{N}}{2} - 1} \exp(-\gamma\tau) d\tau$$

with $\gamma = (\mathring{N}s + 2\beta + \frac{\lambda \mathring{N}(\overline{x}_i - \rho)^2}{\lambda + \mathring{N}})/2$. The term inside the last integral is proportional to the density function of a Gamma distribution of parameters $(\alpha + \frac{\mathring{N}}{2}, \gamma)$. Therefore, Score S_2 is equal to:

$$\begin{aligned} S_2 &= \log \left[\frac{\beta^\alpha \sqrt{\lambda}}{\Gamma(\alpha)(\sqrt{2\pi})^{\mathring{N}+1}}\right] + \log \left[\frac{\sqrt{2\pi}}{\sqrt{\lambda + \mathring{N}}} \frac{\Gamma(\alpha + \frac{\mathring{N}}{2})}{\gamma^{\alpha + \frac{\mathring{N}}{2}}}\right] \\ &= \log \left[\frac{1}{\sqrt{2\pi}^{\mathring{N}}} \frac{\beta^\alpha}{\gamma^{\alpha + \frac{\mathring{N}}{2}}} \frac{\Gamma(\alpha + \frac{\mathring{N}}{2})}{\Gamma(\alpha)} \frac{\sqrt{\lambda}}{\sqrt{\lambda + \mathring{N}}}.\right] \end{aligned}$$

Hence the score displayed in the proposition. ■

4.3 Experiments

In this section, we provide some experiments to highlight the advantages of cdBNs over other models. For this purpose, we compare BNs, ctdBNs and cdBNs on

real-world problems on real-world datasets from the UCI repository [Lichman, 2013] reported in Table 4.1. In these datasets, all rows with missing values were removed. In each dataset, there exists a discrete random variable, call it X_0, representing a classification variable. The other random variables can be either discrete (variables $\mathbf{X_D} = \{X_1, \ldots, X_d\}$) or continuous ($\mathring{\mathbf{X}}_\mathbf{C} = \{\mathring{X}_{d+1}, \ldots, \mathring{X}_n\}$). Our classification problem is to infer the value of X_0 given complete observations for values of variables in $\mathbf{X_D} \cup \mathring{\mathbf{X}}_\mathbf{C}$.

Dataset	#Attr.	#Classes	#Rows	#Cont. attr.
breast	10	2	667	9
cleve	14	2	296	13
crx	16	2	653	6
iris	10	3	150	4
pima	9	2	768	8

Table 4.1: UCI datasets used for BN, ctdBN and cdBN comparisons experiments.

In order to address such a problem with all those three models, we must first discretize all the continuous random variables. We use two different approaches for that:

1. The domain sizes of all the random variables were fixed to the same value and, in different experiments this value was set to 2, 3, 4 and 5. Finally, the discretization intervals were determined in order to maximize the entropy of each continuous random variable (this corresponds to intervals defining uniform distributions).

2. Friedman's discretization algorithm [Friedman and Goldszmidt, 1996].

From that discretization, the original dataset $\mathring{\mathcal{D}}$ is mapped into the fully discrete one $\mathcal{D}$. A BN $\mathcal{B}$ over $(X_0, X_1 \ldots, X_n)$ is then learnt from $\mathcal{D}$ using a hill climbing algorithm with an MDL score [1]. This BN is then exploited for a classification task as follows: given some observation $e_{\mathring{X}_i}$ (resp. e_{X_i}) on each continuous random variable $\mathring{X}_i$ (resp. discrete variable X_i), we enter belief $P(e_{\mathring{X}_i}|X_i)$ (resp. $P(e_{X_i}|X_i)$) into Bayes net $\mathcal{B}$, and then we compute the posterior distribution $P(X_0|e_{X_1}, \ldots, e_{X_d}, e_{\mathring{X}_{d+1}}, \ldots, e_{\mathring{X}_n})$ by means of a Lazy-Propagation inference. The most probable value for class variable X_0 becomes simply:

$$x_0^* = \underset{X_0}{\text{Argmax}}\, P(X_0|e_{X_1}, \ldots, e_{X_d}, e_{\mathring{X}_{d+1}}, \ldots, e_{\mathring{X}_n}).$$

[1] For all Bayesian Network's related learning and inference processes, we used the aGrUM library (`http://agrum.lip6.fr`)

For a fair comparison, we construct cdBN (resp. ctdBN) models as follows: we start from BN $\mathcal{B}$ computed in the preceding paragraphs and we add to it some conditional densities (resp. conditional truncated densities) $h_i(\mathring{X}_i|X_i)$, $i = d+1,\ldots,n$. The latter are determined using a weighted kernel density estimation[2](resp. kernel Density Estimation) with a Gaussian kernel and Scott's rule for estimating the bandwidth [Scott, 1992]. Then the h_i's are i renormalized so that their integrals over the whole domain of each continuous variable (resp. over each interval of discretization) are equal to 1.

The same evidence e_{X_i} and $e_{\mathring{X}_i}$ as those of the BN are entered into the cdBN (resp. ctdBN). However, the latter are included into the cdBN (resp. ctdBN) as beliefs $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ as cdBNs (resp. ctdBNs) can cope with more precise evidence than mere beliefs $P(e_{\mathring{X}_i}|X_i)$ about discretized random variables X_i. Therefore, after entering evidence, the cdBN (resp. ctdBN) represents:

$$g(X_0,\ldots,X_n,\mathring{X}_{d+1},\ldots,\mathring{X}_n,e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n}) = P(X_0|\mathbf{Pa}(X_0))\prod_{i=1}^{n}P(X_i|\mathbf{Pa}(X_i))\times \prod_{i=1}^{d}P(e_{X_i}|X_i)\prod_{i=d+1}^{n}h_i(\mathring{X}_i|X_i)f_i(e_{\mathring{X}_i}|\mathring{X}_i).$$

From this distribution, we perform an inference process to compute the posterior distribution $g(X_0|e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n})$ so the most probable value for class variable X_0 is $x_0^* = \text{Argmax}_{X_0}\, g(X_0|e_{X_1},\ldots,e_{X_d},e_{\mathring{X}_{d+1}},\ldots,e_{\mathring{X}_n})$.

Finally, to perform our experiments, each dataset of Table 4.1 is split into 5-folds, in order to perform cross-validation. All continuous variables are normalized into a $[0,1]$ interval. After learning, for each row of each test set, we estimate the most probable value of class variable X_0 given the observation of the values of $X_1,\ldots,X_d,\mathring{X}_{d+1},\ldots,\mathring{X}_n$ on that row: for the BN, the value of $\mathring{X}_i$ is translated into the corresponding value of its discretized counterpart X_i and $P(e_{\mathring{X}_i}|X_i)$ is precisely the evidence that X_i has taken this value; for the cdBN (resp. ctdBN), belief $f_i(e_{\mathring{X}_i}|\mathring{X}_i)$ is expressed as a normal distribution whose mean is the observed value $\mathring{X}_i$ in the row of the test set and whose variance is arbitrarily set to 0.01. The evidence on discrete variables X_i are entered as classical beliefs $P(e_{X_i}|X_i)$. Once

[2] The weight for each sample is determined by the following equation:

$$w(\mathring{x}_i|X_i) = \begin{cases} e^{k(\mathring{x}_i - t)}, & \text{if } \mathring{x}_i \text{ is not in the interval of } X_i \\ 1, & \text{otherwise} \end{cases}$$

where t represents one of the cutpoints that defines the discretization interval X_i and which is the closest to x_i, and $k = 1$ for all the presented experiments.

the most probable values for X_0 have been inferred from each model, we compare them to the actual value of X_0 observed in the test set. Each model's classification accuracy is computed as the proportion of correct estimations performed.

These accuracies are displayed in Tables 4.2, 4.3, 4.4, 4.5 and 4.6, constructed from the 25 classification experiments executed for each model (5 datasets, using 5 different discretizations for each).

Dataset	**%Acc BN**	**%Acc ctdBN**	**%Acc cdBN**
breast	96.92+-0.55	97.22+-0.56	97.22+-0.56
cleve	80.53+-5.33	81.16+-3.51	81.16+-3.51
crx	84.56+-4.90	86.39+-3.16	86.39+-3.16
iris	94.67+-2.67	95.33+-1.63	95.33+-1.63
pima	73.83+-3.12	73.95+-2.34	74.08+-2.92

Table 4.2: Classification results for discretizations with Domain Size = 2.

Dataset	**%Acc BN**	**%Acc ctdBN**	**%Acc cdBN**
breast	96.85+-0.73	97.14+-0.52	97.14+-0.52
cleve	80.34+-4.13	80.31+-3.04	80.31+-3.04
crx	84.78+-4.15	86.39+-3.16	86.17+-3.53
iris	91.00+-6.16	93.67+-3.14	94.00+-3.27
pima	74.15+-3.45	74.02+-2.38	74.15+-2.61

Table 4.3: Classification results for discretizations with Domain Size = 3.

Dataset	**%Acc BN**	**%Acc ctdBN**	**%Acc cdBN**
breast	96.88+-0.77	97.17+-0.66	97.17+-0.66
cleve	79.71+-3.95	80.60+-3.14	80.60+-3.14
crx	85.06+-3.71	86.39+-3.16	86.24+-3.41
iris	90.44+-5.56	94.22+-3.33	94.44+-3.37
pima	74.26+-3.03	74.08+-2.35	74.08+-2.47

Table 4.4: Classification results for discretizations with Domain Size = 4.

From these tables, we can observe:

- ctdBN performs better or equal than BN 21/25 of times.
- cdBN performs better or equal than BN 22/25 of times.
- ctdBN performs strictly better than BN 15/25 of times.

Dataset	%Acc BN	%Acc ctdBN	%Acc cdBN
breast	97.00+-0.80	97.18+-0.63	97.18+-0.63
cleve	80.16+-3.85	81.17+-3.42	81.17+-3.42
crx	85.13+-3.64	86.39+-3.16	86.28+-3.35
iris	91.17+-5.30	94.50+-3.03	94.67+-3.06
pima	74.38+-3.01	74.38+-2.54	74.41+-2.62

Table 4.5: Classification results for discretizations with Domain Size = 5.

Dataset	%Acc BN	%Acc ctdBN	%Acc cdBN
breast	97.51+-1.20	97.22+-0.86	97.22+-0.86
cleve	79.83+-3.65	82.52+-4.74	82.52+-4.74
crx	85.94+-3.96	86.39+-3.16	86.39+-3.16
iris	92.67+-3.27	96.00+-1.33	96.00+-1.33
pima	72.40+-2.40	73.56+-2.20	73.56+-2.39

Table 4.6: Classification results using Friedman's discretization.

- cdBN performs striclty better than BN 16/25 of times.
- cdBN and ctdBN perform similarly 15/25 of times.
- ctdBN performed strictly better than cdBN only 3 times.
- cdBN performed strictly better than ctdBN only 7 times.

We can therefore conclude that, as expected, that the presented cdBN model is better suited than BNs for performing classification tasks, and even slightly better ctdBNs for that task.

4.4 Conclusion

In this chapter, we have introduced a new graphical model called *Conditional Densities Bayesian Networks*. This one is a generalization of the ctdBNs presented in the preceding chapter. It can be considered as a robust version of ctdBNs w.r.t. discretizations. We have shown that this new model has attractive properties, notably in terms of expressiveness, i.e., it can approximate most of the usual mixed probability distributions. We have also detailed how inference can be performed in cdBNs and provided some complexity results. We have also described how structure learning can be performed using an SEM-like algorithm and, for this purpose, we have introduced a new scoring function well-suited to cdBNs. Finally, some

experiments have been performed to compare the effectiveness of cdBNs against ctdBNs and classical BNs on classification tasks.

Chapter 5

SCISSOR project

This thesis was funded as part of the H2020[1] European project SCISSOR[2], which proposes a new SCADA[3] security monitoring framework. This one aims to meet the requirements for dealing with critical infrastructures[4], while relying on open-source technology and open standards; this chapter is dedicated to the real-world application of the previously presented models, in particular cdBNs. In Figure 5.1, we present an overly simplified schema of how our cdBN module interacts with a SCADA System: The cdBN module gets real-time (sensors) measurements and uses them in batch to learn at regular intervals a new cdBN model. At the same time, exploiting the last cdBN model learnt, each time it receives some new measurements (data), it performs a likelihood-based anomaly detection. Whenever the current system state appears to be unlikely, i.e., it has a too low likelihood, our anomaly detection module sends an alert to the system because this situation might represent a cyber-attack attempt or a misuse of the system. The SCISSOR framework (its technologies and tools) has been successfully validated both on an off-line realistic SCADA platform and in the real-world testbed trials in SEA FAVIGNANA[5]. These validations include applications of the cdBN model presented in this thesis.

The rest if this chapter is organized as follows: Sections 5.1 and 5.2 are devoted to give an overview of the SCISSOR project and to the cdBN Threat detection module, *i.e* how our work coexist with the rest of the project. Section 5.3

[1] `https://ec.europa.eu/programmes/horizon2020/en/`

[2] European project H2020-ICT-2014-1 #644425. For further information, refer to `https://scissor-project.com/`

[3] SCADA is a large-scale remote management computer system for real-time processing of telemetry and remote control of technical installations.

[4] Critical infrastructures are essential assets for a society and economy, *e.g.* water supply, transportation, electricity generation, etc.

[5] Favignana Island's Electric Central (Sicily region, Italy), ran by SEA (Società Elettrica di Favignana). We will refer to it simply as *Favignana*.

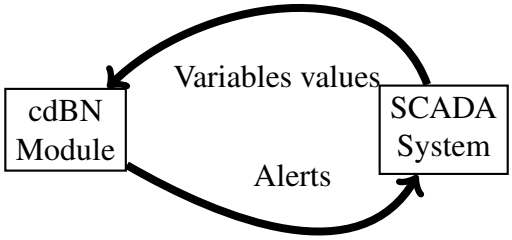


Figure 5.1: Simplified schema

gives a comparison between a cdBN temporal model and the Kalman filter in the context of anomaly detections, which were made offline (*i.e.* with recorded data, not in real time). this comparison was made at request of H2020 reviewers domain experts. Finally, Section 5.4 presents the two sets of experiments included in the SCISSOR testbed.

5.1 Project description and scope

The SCISSOR project proposes a holistic, end-to-end, multi-layered security monitoring framework, which includes encryption of the communications and access control support. This framework focuses on the SCADA system security but it is extensible to include many supplementary sensing, monitoring, and security assessment modules *i.e.* the SCISSOR architecture is meant to be incrementally deployable, and to provide a unified way to handle, control, and correlate widely heterogeneous remote monitoring sources and relevant data.

The SCISSOR framework is structured into four layers, and its simplified view is depicted in Figure 5.2. From bottom to top, it is composed of a Monitoring Layer (ML), a Control & Coordination Layer (CCL), a Decision & Analysis Layer (DAL) and a Human Machine Interface (HMI).

The Monitoring Layer (ML) integrates multi-source, multi-technology, and multi-purpose (environmental, traffic, system, surveillance) monitoring and sensing technologies and devices. The analysis of the information produced by the ML elements allows the SCISSOR system to identify security issues. The ML sends its information to the Control & Coordination Layer (CCL). The CCL can issue commands to the ML elements to configure their monitoring operations and in some cases to react to detected events.

The SCISSOR architecture is open to integrate any type of monitoring elements, but the design and implementation of the ML in the project has focused on

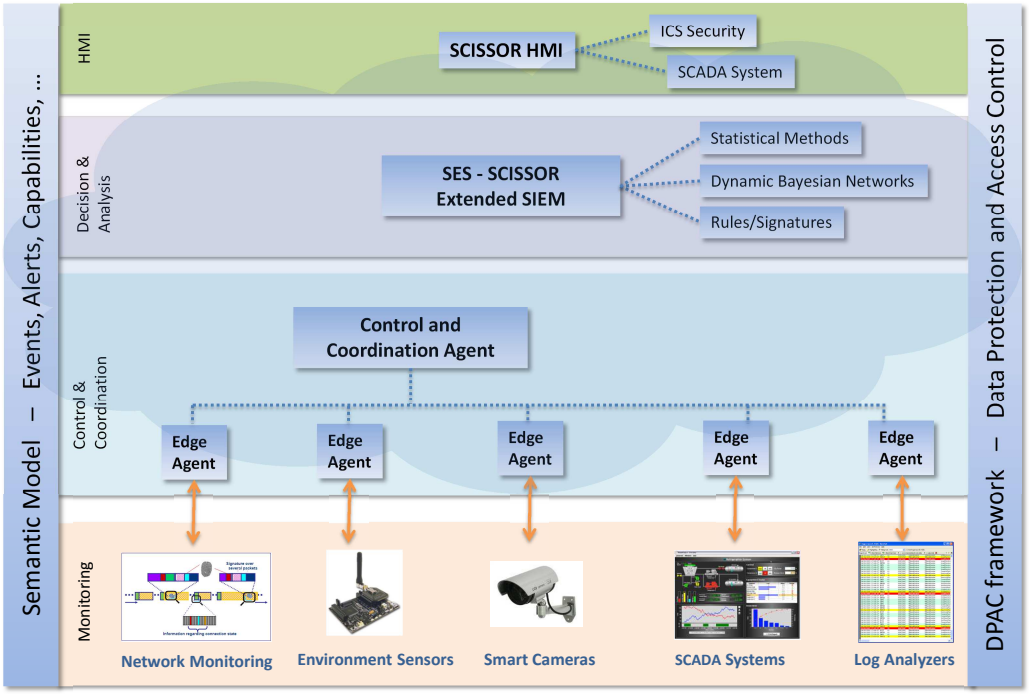


Figure 5.2: Overall architecture of SCISSOR system

the following components, that are included in the SCISSOR testbeds:

- Network monitoring
- Environment sensing
- Smart cameras
- ICS/SCADA[6] monitoring
- Various logs from applications, firewalls, intrusion detection systems, etc.

The Control & Coordination Layer (CCL) processes and enriches the information coming from the monitoring components. It cryptographically protects and delivers the gathered signals and events to the Decision and Analysis Layer (DAL). The CCL allows the DAL to operate on normalized information, by dealing with the different formats and interaction modes of the monitoring components. It also supervises the operations of the monitoring elements by sending configuration commands (*e.g.* activation, de-activation). It can autonomously react to some detected events by sending proper commands. The CCL provides the

[6]ICS = Industrial Control Systems.

DAL with an abstracted view of the monitoring components. A set of semantic models supports the operation of the CCL and it is used in particular to enrich the information sent towards the DAL. The CCL is composed of:

- Multiple Edge Agents (specialized per monitoring element type).
- A logically central Coordination and Control Agent.

The Decision & Analysis Layer (DAL) processes all the information coming from the CCL. It is in charge of correlating events, detecting behavioral changes in the system as a whole, and committing mitigation decisions. The DAL extends the existing SIEM (Security Information and Event Management) approach and implementations with advanced algorithms operating on the information gathered by heterogeneous set of monitoring components, enriched and normalized by the CCL. The DAL integrates advanced analysis approaches, including robust statistical methods and inference of BN-based models (notably the cdBN model, presented in this thesis).

The human-machine interface (HMI) layer is devised to present "behavioral phenomena" (rather than raw data) to the human end user in a simple and usable manner.

A realistic deployment of the SCISSOR system potentially needs to process in real-time or near-real time a large amount and variety of data, whose sources can be highly distributed. The monitoring and logging processes are naturally distributed, so the need for rapid analysis suggests that the computing resources must be distributed as well. The load on the system, especially in response to attacks or unseen system behaviors, will vary greatly over time. In short, SCISSOR requires a distributed, agile computing platform to provide the computing power necessary to counter potentials attacks on a SCADA system. These requirements are fulfilled with the integration of cloud computing technologies, in particular hybrid cloud computing technologies.

5.2 CdBN threat detection module

The DAL is composed of a classical SIEM[7] gathering, processing and storing events from the lower layers, enhanced with new analysis modules using dynamic cdBN inferences and robust statistical methods. The overall interactions among these components are illustrated in Figure 5.3. The SCISSOR Extended SIEM (SES, on the left side of the figure) will be responsible for the interactions between the *Decision and Analysis (DAL)* layer and both the *Control and Coordination (CCL)* and the Human Machine Interface (HMI) layers.

[7]SIEM = Security information and event management.

The DAL can hold several Threat Detection Modules (TDM) whose purpose is to add thorough analysis of messages transiting through the SCISSOR Messaging Infrastructure (SMI). Such messages are composed of logs (from sensors, servers, etc.) and message using the Intrusion Detection Message Exchange Format (IDMEF) [Debar et al., 2007].

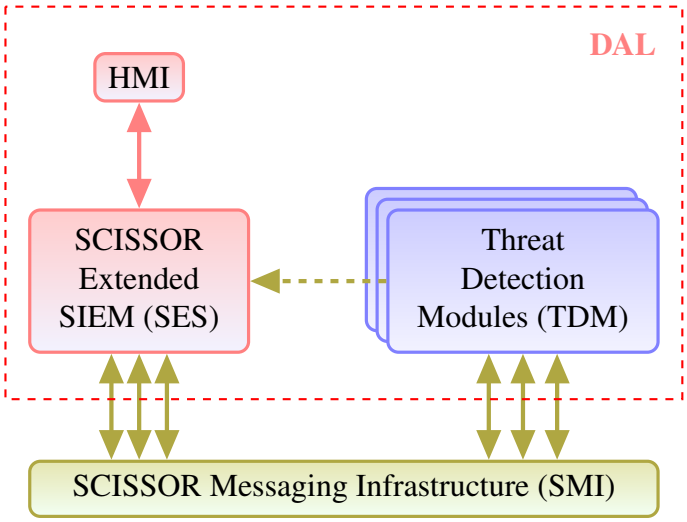


Figure 5.3: Interactions among the Decision and Analysis Layer components

The SES sends to the HMI the data needed by the human operator to visualize the current state of the system. As we can see in Figure 5.3, neither the HMI nor the SES directly interact with the TDM, relying instead on the IDMEF messages emitted by them. Using a standard such as IDMEF also helps interaction among the TDM as they will be able to consume others modules messages.

CdBN TDM architecture

The cdBN TDM (BN-TDM) is organized as a set of services implemented in python, C++14 and/or shell. These services are described in Figure 5.4.

As shown in Figure 5.4, the messages sent through the SMI are first consumed by a Kafka consumer written in Python3. This consumer then sends the message to a parser/discretizer, also written in Python3. The latter analyzes the message in order to extract the features of interest but also to discretize the continuous features that need be discretized in the Bayesian network-based model. As an example, in the ASSYSTEM testbed, many information sent by the SCADA system are in practice real numbers due, for instance, to noise measurements, but they should be interpreted as discrete, if not binary, values. The discretizer transforms such real

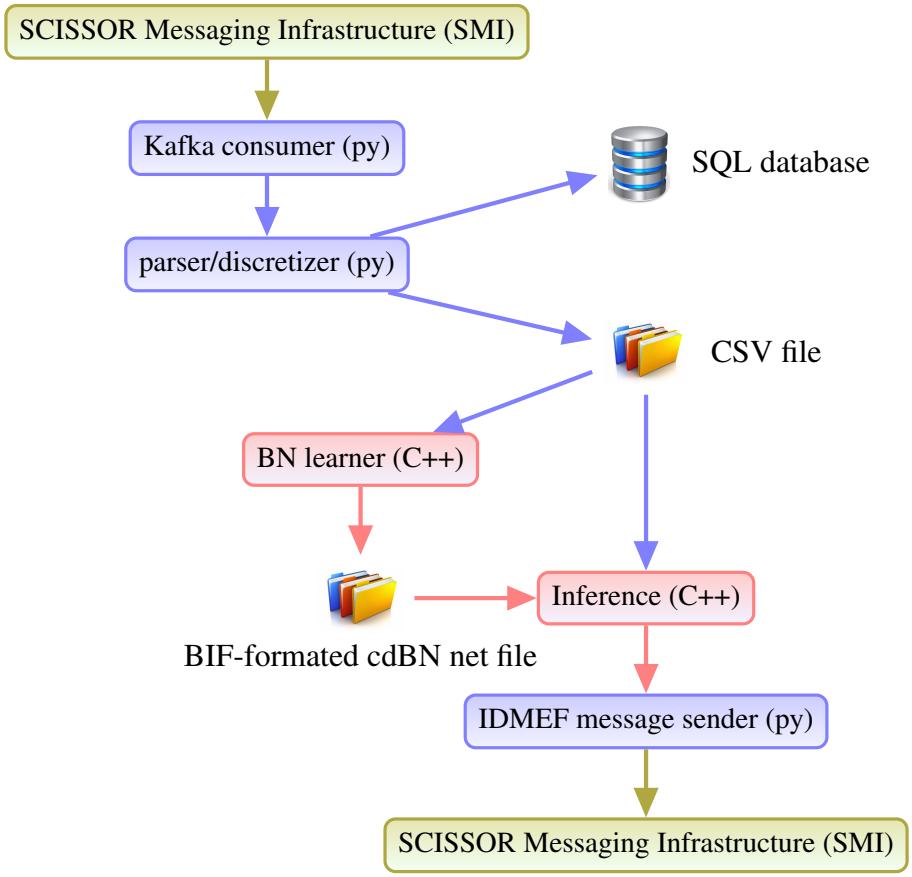


Figure 5.4: The architecture of the cdBN TDM.

values into their appropriate discrete counterpart. Such a mapping is necessary to reduce the learning phase (discrete variables and their relationships are indeed much simpler to learn than continuous variables and their relationships). The data resulting from the parser/discretizer can be stored into a database, either a SQL one or a CSV file. Currently, we exploit a CSV file. The purpose is twofold: first, the robust statistics TDM takes a CSV file as input, so the file created by the cdBN TDM can be reused by the other modules or, at least, if both TDMs are run on different virtual machines, the same program can be used to generate the databases used by both TDMs[8]. Second, the virtual machines executing the Bayesian network TDM are not very powerful (their amount of memory and their CPU are limited). So, using a CSV file reduces the overhead induced by a SQL database server. However, when using more powerful computers, the Bayesian

[8]In fact there is also a Robust Statistics TDM model running with the cdBN TDM, sharing the exact same data.

network TDM can substitute the CSV file by a SQL database in a transparent way for the user. The "old" data stored in the database are exploited twice a day by a BN learner to build/update the mathematical model. The "new" data received are processed immediately by a BN inference mechanism to detect unusual situations. In this case, an IDMEF message is constructed by a Python3 script and it is sent to the SMI, so that both the SIEM and the other TDMs can be aware of it.

The cdBN TDM implementation is split into different directories as follows:

- **shared**: this directory contains the parameters used by the SMI data consumer, the learning and the inference engines. Notably, the parameter file contains keywords included in the SMI messages that determine to which model they apply: actually, it is inefficient to construct a single probabilistic model to deal with the whole SCADA system. For instance, Favigana's data have nothing to do with those of the ASSYSTEM testbed, so why should we construct a single model for both testbeds? In addition, even in Favignana, all the electric lines should be dealt independently because their data arrive at different times. As a consequence, keywords FAVIGNANA_Q_L3_1_MIN, FAVIGNANA_Q_L2_1_MIN, FAVIGNANA_Q_L1_1_MIN, FAVIGNANA_P_L3_1_MIN, FAVIGNANA_P_L2_1_MIN and FAVIGNANA_P_L1_1_MIN that indicate to which electric line the SMI message is related, are exploited by the data consumer to produce different CSV files and, therefore, different probabilistic models. All the keywords that determine the different models to use are specified in the parameter files of the *shared* directory. The CSV files as well as the probabilistic models are named according to these keywords and are all stored in the *shared* directory.

- **data_consumer**: this directory contains the Python3 script that subscribes to kafka and consumes the data. The same script parses the data collected and transforms them in order to simplify the tasks of the mathematical modules. Finally, the script organizes these data so that they can be stored into CSV files and launches periodically (every 12 hours) a learning process. The data consumer is exclusively written in Python3. It is intended to be executed at the startup of the virtual machine and its execution lasts until its shutdown.

- **learning_engine**: this directory contains the C++14 implementations of the learning algorithms to learn cdBN models. They rely on the C++14 open source aGrUM library. These algorithms construct from the data stored in the CSV files located in the *shared* folder the cdBN mathematical models used for threat detection. They store their resulting models in extended

BIF-like format. The C++14 learning program is executed from a Python3 wrapper which is called every 12 hours by the data consumer script.

- **inference_engine**: this directory contains the C++14 implementations of the inference algorithms used to detect threats as well as their Python3 wrappers. Using the models learnt by the *learning engine* scripts, the inference engine consumes data from the SMI and computes the posterior probabilities given the evidence provided by the consumed data. Then, based on these probabilities, they determine whether to send an alarm or not. To easily interface the inference engine with the consumed data and with the process to send alarms, the C++14 program is accessed through a Python3 wrapper. The latter encodes the messages in IDMEF format. As for the learning engine and the data consumer, the Python3 script of the inference engine reads its configuration from the parameters file located in the *shared* folder and executes the models to which the new data received from the SMI apply. The inference engine script is executed at the startup of the virtual machine and its execution lasts until its shutdown.

5.3 Comparison between the Kalman filter and non-stationary cdBNs in anomaly detection.

In this section, we present some experiments performed on some Favignana's testbed data sent through the SCISSOR architecture and captured by the DAL. Here, the goal is to assess whether our models are able to detect anomalies or not. For this purpose, we store Favignana's data into the CSV database designed for the DAL models and extract from it some parts to learn the models, the rest being used for tests. In the latter, we generate manually some anomalies, as shown below. The models tested are a Kalman filter, which was a requirement of the H2020 project reviewers, and non-stationary cdBNs: in Section 1.7, we presented non-stationary DBNs as piecewise stationary DBNs, i.e., as sets of pairs $\langle(\mathcal{B}_h, T_h)\rangle_{h=0}^{m}$ of BN fragments and transition times. CdBNs are defined in a similar fashion, substituting BN fragments by cdBN fragments.

5.3.1 Data source and preprocessing treatments

The data on which experiments are performed were sent over the SCISSOR's VPN and transmitted through the SCISSOR's messaging infrastructure (SMI) over a period of seven days in March 2017. Data were collected from the Favignana's substations on a rate of approximately one message every two seconds.

For learning, these data require preprocessing for five reasons:

1. first, some columns of the database correspond to "raw" json metatadata or correspond to folder names, integer IDs, *etc.* and have to be excluded from the experiments because they are irrelevant. This is not an issue because such information can be defined when designing the messages to be sent over the SMI. Note that IDs may be of interest when raising IDMEF alarms to inform the end users of which part of the system or which record is involved in the alarm. But the precise ID itself is useless to determine whether an alarm should be raised or not. This explains why it should not be included into the probabilistic distribution (it should just be included in the alarm message).

2. second, sometimes, the data retrieved contain spurious values. These can be "huge" and meaningless values, often caused by network communication issues resulting from substations being temporarily unable to send the information of their sensors. In turn, the result is that the SCADA system is unable to receive data from PQA[9] and, as a consequence, the load flow algorithm is unable to perform correct evaluations and it produces unusually high numbers.

   Per se, such spurious values do not represent a major problem for learning cdBNs. One only needs to take care to initialize the SEM-based learning algorithm with a value of the latent variables representing these spurious values. This is easily guaranteed by sorting the values of each database's variable in increasing order, by splitting the domain size of these variables into equal frequency intervals and by assigning the records' values to the corresponding intervals.

   On the contrary, for the Kalman filter, these spurious values need absolutely be removed. Actually, Kalman filters represent unimodal probabilistic models; and as such, they are unable to cope with multiple local maxima. But, in the Favignana's dataset, "correct" variables' values form one distribution with its own mode and the spurious values form another distribution with another very distinct mode, the overall distribution being a mixture of these two. As shown in the experiments below, when data have several local maxima, Kalman is often unable to detect anomalies. By not removing spurious values, the resulting Kalman filter is unable to detect any anomaly.

   So, in order to compare cdBNs and Kalman filters, we need to remove the spurious values. To do so, we substitute them by the last "correct" values observed for the variables.

3. The third issue we encounter to model the probabilistic models results from

[9] PQA = Power Quality Analyzer

the existence of linear dependences between some random variables. Again, this is not really an issue for learning cdBNs, although learning is somewhat more difficult in this case (see [Mabrouk et al., 2015] for rules to learn cdBN models in the presence of deterministic relationships between some random variables). The issue essentially concerns the Kalman filter. Broadly speaking, some steps in the learning of this filter require some matrix inversion that cannot be done when there exist linear dependences. In addition, the values of some variables are constant in the whole database and this also raises some learning issues. So we remove them as well. The list of removed variables is the following one:

- Q_INTERMEDIA2
- Q_L252
- FP_L1333
- FP_L1211
- P_L1_OR333
- P_L1_OR324
- P_L1_OR322
- FQ_L1211
- Q_L1_OR322
- Q_L1_OR325
- Q_L1_OR333
- P_L252
- P_INTERMEDIA2

As a result, the final number of variables (columns in the database) is equal to 69.

4. In the database, some values are missing. This is due to the data being sent in an asynchronous manner (recall that each record corresponds to a tuple of the values of all the variables). Asynchronicity makes the variables unobserved for very small amounts of time (usually about 2 seconds). Therefore, it is meaningful to substitute them by the last observed values.

5. For the purposes of this comparison (due to Kalman's filter limitation) all discrete variables are systematically ignored.

Finally, all the records are sorted by increasing time of day (hh:mm:ss). This enables to compute the temporal evolution of the data, which is the core of the Kalman filter but also of the non-stationary cdBNs. The data are also normalized in such a way that, for every column, the average value is set to 0 and its variance set to 1. This final transformation is not necessary for learning but makes it easier to display the results of the experiments, and enhances numerical precision in the computations.

5.3.2 The Kalman Filter's parameter learning

Here, in order to be self-content, we briefly recall the basics of Kalman filtering. The Kalman filter is an algorithm designed to estimate a joint probability distribution over time of a (possibly multidimensional) process whose state variable X is hidden to the user but is observable through a (possibly multidimensional) variable Y. The state space representation of the process is the following:

$$\begin{aligned} X[t+1] &= \mathbf{A}X[t] + \mathbf{B}U[t] + \mathbf{Q}\omega \\ Y[t] &= \mathbf{C}X[t] + \mathbf{D}U[t] + \mathbf{R}\nu \end{aligned}$$

where:

- t represents the current time step.
- X represents the current (hidden) multidimensional state of the process.
- Y represents the observations we have on the state of the process (the sensor lectures).
- U represents the control inputs of the process (equal to zero, in our particular case).
- $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are matrices defining the parameters describing the relationships between the random variables and the dynamics of the process.
- $\mathbf{Q}$ and $\mathbf{R}$ are covariance matrices representing the uncertainties of the current state and the current observations respectively.

Let $\hat{x}$ and $\hat{y}$ denote the expected values of X and Y respectively. The Kalman filter estimates the values $\hat{x}$ and $\hat{y}$ at each time step using the following equations:

$$\begin{aligned} \hat{x}[t+1] &= (\mathbf{A} - \mathbf{KC})\hat{x}[t] + (\mathbf{B} - \mathbf{KD})u[t] + Ky[t] \\ \hat{y}[t] &= \mathbf{C}\hat{x}[t] + \mathbf{D}u[t] \end{aligned}$$

where $\mathbf{K}$ is also a matrix, known as a *Kalman gain*. For learning the Kalman filter's parameters ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{K}$, $\mathbf{Q}$ and $\mathbf{R}$), we use the *n4sid* function from *GNU-Octave*'s control package [10].

5.3.3 Anomaly detection using the Kalman filter

The Kalman filter assumes that each observation variable $Y[t]$ follows a multidimensional normal distribution with mean $y[t]$ (the value actually observed) and covariance $\mathbf{R}$, which allows to define the log-likelihood of the observation as:

$$LL(y[t]) = -\frac{p}{2}\log 2\pi - \frac{|\mathbf{R}|}{2} - \frac{1}{2}y[t]^T\mathbf{R}^{-1}y[t]$$

where p is the dimension of each observation vector $y[t]$. For detecting anomalies, it could be sufficient to exploit the value $LL(y[t])$ in a rule such as "if $LL(y[t])$ is below a given threshold, then this corresponds to an anomaly, so raise an alarm". But, as suggested in "*Adaptive Kalman Filtering for Anomaly Detection in Software Appliances*" by Knorn and Leith, it would be better to avoid log-likelihood instability, i.e., to prevent its value ever shifting between high and low values. For this purpose, we use a lowpass filter on that value defined as follows:

$$z[t] = \alpha z[t-1] + (1-\alpha)LL(y[t]),$$

with $\alpha \in [0, 1]$. In our experiments, we use $\alpha = 0.5$, which conveniently avoids many false positive anomaly detections. We define a threshold value such that for every time t when there is an anomaly, the following inequality should be satisfied:

$$z[t] < threshold.$$

5.3.4 Anomaly detection using non-stationary cdBNs

To properly learn the non-stationary cdBN model, we automatically add new features to the already preprocessed data: for each continuous variable $\mathring{X}$, we add a new variable $\mathring{X}_{diff}$ that represents its evolution over time. It is computed as:

$$\mathring{X}_{diff}[t] = \mathring{X}[t] - \mathring{X}[t-1]$$

The learning dataset is also split into epochs in order to cope with the non-stationarity of the process. In each epoch h, a cdBN B_h is learnt. We exploit such

[10] `https://octave.sourceforge.io/control/function/n4sid.html`

cdBN on the test dataset, considering that there exists an anomaly whenever the log-likelihood w.r.t. the cdBN of the observation is below a given threshold, i.e.:

$$LL_{B_h}(observation) < threshold,$$

where $LL_{B_h}(\cdot)$ represents the log-likelihood of a given observation in epoch h. The threshold is set to $\mu - 6\sigma$, where μ and σ are the mean and the standard deviation respectively of the set of posterior log-likelihoods of all the samples used to learnt the cdBN.

5.3.5 Protocol of the experiments

We perform the anomaly detection tests in a two-step process: first, we learn both models (the Kalman filter and the non-stationary cdBN), and, then, those are exploited for anomaly detection on the same test datasets.

Following Favignana's domain experts, we define the following 7 transition times:

$$\text{1h30, 3h30, 6h30, 8h30, 9h30, 18h30, 23h30.} \tag{5.1}$$

Once the transition times are known, we split the whole Favignana's dataset $\mathring{\mathcal{D}}$ into datasets $\mathring{\mathcal{D}}_h$ corresponding to the 8 epochs h resulting from these transition times (see Eq. (5.1)). Finally, each dataset $\mathring{\mathcal{D}}_h$ is further split into a learning dataset $\mathring{\mathcal{D}}_h^L$ and a test dataset $\mathring{\mathcal{D}}_h^T$ (70% of $\mathring{\mathcal{D}}_h$ being devoted to $\mathring{\mathcal{D}}_h^L$ and the 30% remaining to $\mathring{\mathcal{D}}_h^T$). In each epoch h, the final cdBN $\mathcal{B}_h$ used in the anomaly detection tests are learnt from $\mathring{\mathcal{D}}_h^L$.

The Kalman filter corresponds to a stationary model, hence epochs are irrelevant for it. Therefore, we learn it from the whole Favignana's dataset $\mathring{\mathcal{D}}$. Note that, to be more fair with cdBNs, for testing the Kalman filter on the test dataset $\mathring{\mathcal{D}}_h^T$, we should remove from learning dataset $\mathring{\mathcal{D}}$ all the records of $\mathring{\mathcal{D}}_h^T$ because this advantages the Kalman filter over the non-stationary cdBN model since part of its parameters are tailored using the test dataset. However, for simplicity, we chose not to remove $\mathring{\mathcal{D}}_h^T$ from $\mathring{\mathcal{D}}$ when learning the Kalman filter's parameters.

After the completion of the above processes, 8 cdBNs $\mathcal{B}_h$, $h = 0, \ldots, 7$, are available, as well as one Kalman filter and 8 datasets $\mathring{\mathcal{D}}_h^T$. The latter are called "datasets without perturbations". In addition to these datasets, for each epoch h, we have constructed 6 "perturbed" datasets denoted $\mathring{\mathcal{D}}_h^{T(i)}$, $i = 0, \ldots, 5$, defined as follows: we first select a time interval $[t_0, t_f]$ (see below). Then, for each record of the unperturbed dataset $\mathring{\mathcal{D}}_h^T$ observed in this time interval, we add to the values of the record a value $\epsilon(t)$ defined as:

$$\forall t \in [t_0, t_f],\ \epsilon(t) = K \sin \frac{(t - t_0)\pi}{(t_f - t_0)}.$$

Therefore, the magnitude of the perturbation changes over time. Time intervals $[t_0, t_f]$ are defined as follows:

- Dataset $\mathring{\mathcal{D}}_h^{T(0)}$: Interval $[t_0, t_f]$ corresponds to the whole dataset $\mathring{\mathcal{D}}_h^T$.
- Dataset $\mathring{\mathcal{D}}_h^{T(1)}$: Interval $[t_0, t_f]$ corresponds to the first third of Dataset $\mathring{\mathcal{D}}_h^T$. Recall that the observations are sorted by increasing time order. Hence, only the observations that occurred first in the time series are affected by perturbations, those occurring after t_f remain unaffected.
- Dataset $\mathring{\mathcal{D}}_h^{T(2)}$: Interval $[t_0, t_f]$ corresponds to the second third of Dataset $\mathring{\mathcal{D}}_h^T$.
- Dataset $\mathring{\mathcal{D}}_h^{T(3)}$: Interval $[t_0, t_f]$ corresponds to the last third of Dataset $\mathring{\mathcal{D}}_h^T$.
- Dataset $\mathring{\mathcal{D}}_h^{T(4)}$: Interval $[t_0, t_f]$ corresponds to the first two thirds of Dataset $\mathring{\mathcal{D}}_h^T$.
- Dataset $\mathring{\mathcal{D}}_h^{T(5)}$: Interval $[t_0, t_f]$ corresponds to the last two thirds of Dataset $\mathring{\mathcal{D}}_h^T$.

Finally, for each epoch, and each record of each dataset $\mathring{\mathcal{D}}_h^T$ and $\mathring{\mathcal{D}}_h^{T(i)}$, $i = 0, \ldots, 5$, we compute the log-likelihood estimated by both models (Kalman and cdBNs) and determine whether they can detect anomalies or not.

5.3.6 Results of the experiments

The results obtained on the Favignana's data are summarized in Tables 5.1 and 5.2.

From Table 5.1, we can observe that cdBNs are slightly more prone to produce false positives than Kalman filters. But remember that the latter is advantaged compared to cdBNs since the test database is used when learning the parameters of the Kalman filter: the test and learning databases represent 30% and 70% of the whole database $\mathring{\mathcal{D}}$ respectively. Therefore, about one third of the data used to learn the parameters of the Kalman filter are also used during anomaly detection tests. For cdBNs, the detected anomalies occur in the sixth and seventh epochs (see Figures A.6 and A.7 on Pages 202 and 203 respectively). Note that in both figures, the detection of anomalies is "weak" in the sense that the likelihoods that triggered the anomaly detection are only slightly below the detection threshold. This is also the case for the Kalman filter.

From Table 5.2, it can be observed that non-stationary cdBNs significantly outperform Kalman filters when data contain anomalies. Indeed, they are capable to detect all the anomalies generated, whereas the Kalman filter cannot detect

Table 5.1: Anomaly detection results over unperturbed data

Unperturbed Data	**cdBN model**	**Kalman Filter**
No anomaly detected	6 / 8	7 / 8
Anomalies detected	2 / 8	1 / 8

even half of them. The reasons are twofold, as will be detailed in Subsection 5.3.8: i) the distribution of the data over time is non-stationary, which makes the Kalman filter imprecise since it only models an "average" behavior over time; and ii) all the variables are not unimodal, but the Kalman filter models only unimodal (Gaussian) distributions. This makes it in general not suitable to detect anomalies. Unlike Kalman, non-stationary cdBNs are able to cope with both features and are therefore better suited for anomaly detection.

Table 5.2: Anomaly detection results over perturbed data

Perturbed Data	**cdBN model**	**Kalman Filter**
No anomaly detected	0 / 48	23 / 48
Anomalies detected	48 / 48	25 / 48

5.3.7 Detailed results

The detailed results of this experiments can be found in the Appendix A.

5.3.8 Analysis of the results

As can be observed in the detailed plots in the appendix, in many situations, the Kalman filter is unable to detect anomalies while the non-stationary cdBN can detect them. The analysis of the Favignana's input dataset provides some hints to support this observation. First, the temporal data follow a non-stationary distribution. As there are numerous variables in the dataset, it is not easy to show the shape of the high-dimensional distribution (there are 69 dimensions). Fortunately, we can observe this non-stationarity even on single variables. For instance, the values of Variable "`FP_L1322`" observed in the dataset are shown in Figure 5.5. It can be seen that the values are essentially constant (up to some noise) up to record 3000 but they vary differently after this record (from 3000 to 3800, the values tend to decrease and, after this, they tend to increase up to record 4800 and, then, they decrease again). As a result, for such data, the Kalman filter tends to be imprecise because, by modeling a stationary process, its learnt parameters

correspond to the best "average" stationary model. So, it is unable to discriminate the constant part (before record 3000) from the decreasing and the increasing parts. Therefore, it is unable to detect anomalies resulting from a sequential increase in the values of `FP_L1322` in the period corresponding to the first 3000 records. Similarly, during the period corresponding to records 3800-4800, the values should increase but the Kalman filter will not raise any alarm if the values are constant because, in its "average" model, this does not seem unlikely.

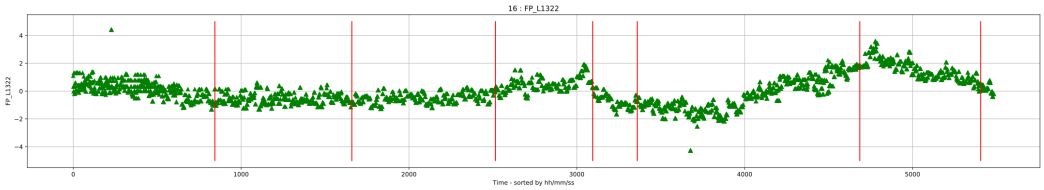


Figure 5.5: The values of Variable "`FP_L1322`".

When the variations due to the non-stationarity are not too high, the Kalman filter provides good results. However, in the dataset, there are not many variables for which this feature holds. Worse, most of the variables are not distributed w.r.t. unimodal distributions. For instance, the values of Variable "`VMT_L21`" as displayed in Figure 5.6 clearly show a bimodal distribution in the period of time between records 700 and 1700. Unlike cdBNs, Kalman filters model only unimodal distributions. Therefore, to best fit the data, the Kalman's parameters correspond to an average between these two modes. To say it differently, the distribution of the data in interval 700-1700 should be represented by a mixture of two unimodal (maybe Gaussian) distributions. This is precisely what the cdBN does in such a case. But the Kalman filter just models a unique Gaussian distribution that tries to best fit this mixture. For this purpose, it is obliged to consider as its mean parameter the average of the means of the two distributions of the mixture and its covariance matrix should contain large numbers so that the observed values do not seem unlikely. As a consequence, observing a sequence of constant values in between the two means of the mixture is never unlikely for a Kalman filter, although this is far from what is observed in the data. This makes the Kalman filter unable to observe many anomalies. On the other hand, cdBNs can detect such anomalies because, as we have seen, they represent mixtures of distributions.

The variables mentioned above are not isolated cases. Figure 5.7 shows other examples of variables for which stationarity and/or unimodality do not hold.

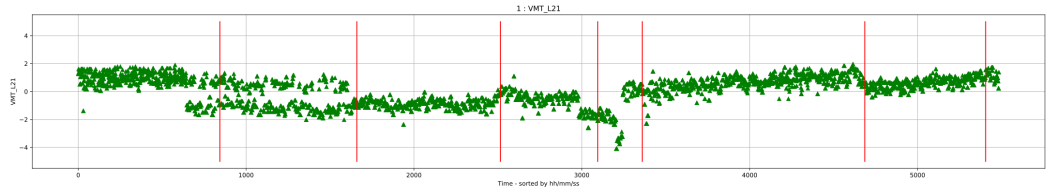
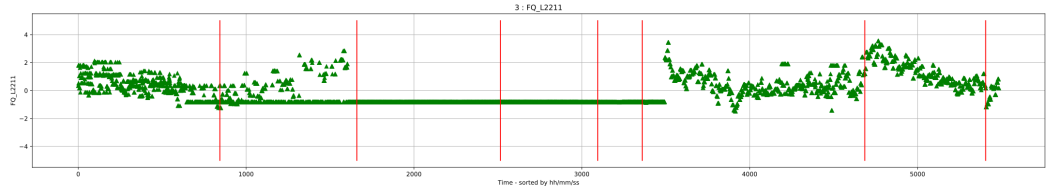
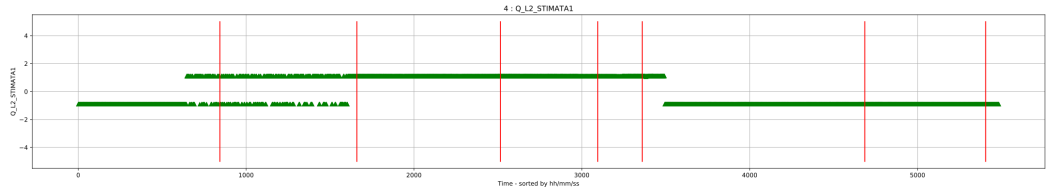


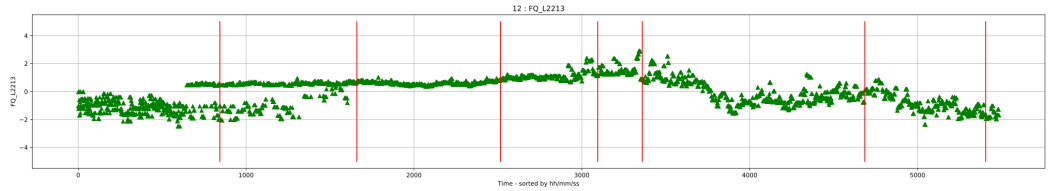
Figure 5.6: The values of Variable "VMT_L21".



The values of Variable "FQ_L2211"



The values of Variable "Q_L2_STIMATA1"



The values of Variable "FQ_L2213"

Figure 5.7: Non-stationarity and non-unimodality in Favignana's data.

5.4 SCISSOR Testbed: cdBN TDM tests.

5.4.1 Monitoring setup

In order to demonstrate the effectiveness of the cdBN threat detection module (TDM), we have performed two sets of experiments on Lines FAVIGNANA_P_L2_1_MIN and FAVIGNANA_P_L3_1_MIN of the Favignana SCADA testbed. For simplicity, hereafter, we call them PL2 and PL3 respectively. As we will notice, the experiments of this section contain much fewer

variables than the ones in the preceding section. This is because the Favignana's expert advised us to create separate cdBN models for each power line (which helps us to overcome the virtual-machine's memory and CPU limitations, and to be able to send alerts in real time). For both experiments, we use observations (messages) transmitted over the SCISSOR messaging infrastructure (SMI) from March 1st, 2018, to March 4th, 2018. In the sequel, we refer to March 1st as "Day 1", to March 2nd as "Day 2", and so on, and we refer to the four days spanning from March 1st to March 4th as "all days".

Each message emitted over the SMI contains many fields, some of which being useless for probabilistic models (e.g., the "event.severity" field, but more generally any field which does not correspond to a variable of the SCADA system). Therefore, within the cdBN TDM, there exists a configuration file that specifies which messages the TDM copes with (the different lines of Favignana are for instance supervised by the cdBN TDM whereas the outputs of the cameras are not), and which fields are of interest. For `PL2` and `PL3`, these fields, which we interpret as random variables, are specified in Tables 5.3 and 5.4 respectively. Note that we also excluded some fields that could have been relevant to the cdBN TDM, when their values were not constant. Indeed, in our test protocol, either such constant values would never be involved in any alarm raising (2nd set of experiments) or they would always raise alarms (1st set of experiments). Therefore, they are not useful to assess the effectiveness of the cdBN TDM.

Variable name
DATE (dd/mm/yyyy)
TIME (HH:MM:SS)
P_L2_OR213
P_L2_OR212
P_L2_OR211
P_L2_OR111
P_L252
VMT_L21
FP_L2213
FP_L2212
FP_L2211
FP_L2111
P_FINALE_W2

Table 5.3: The fields of `PL2` used by the cdBN TDM.

The test protocol used in both experiments is the following: for each day, we learn a dynamic conditional densities Bayesian network (cdBN) from the data

Variable name
DATE (dd/mm/yyyy)
TIME (HH:MM:SS)
P_L3_OR423
P_L3_OR422
P_L3_OR421
P_L3_OR411
P_L3_OR311
P_L3_OR111
P_L353
VMT_L31
FP_L3423
FP_L3422
FP_L3421
FP_L3411
FP_L3311
FP_L3211
FP_L3111
P_FINALE_W3

Table 5.4: The fields of `PL3` used by the cdBN TDM.

received on that day. Let us call these models `cdBN1` to `cdBN4`. In addition, we compute a threshold used to raise alarms: basically, when new data are received by the cdBN TDM, using the cdBN learnt, we compute their posterior log-likelihoods given the last observed data and, when these posteriors are below the threshold, and alarm is raised. As in the preceding set of experiments, the threshold is set to $\mu - 6\sigma$, where μ and σ are the mean and the standard deviation respectively of the set of posterior log-likelihoods of all the samples used to learnt the cdBN. Finally, each cdBN thus constructed is exploited to detect anomalies over a 24 hour set of test observations. The two sets of experiments just differ in the way these observations are defined.

5.4.2 First set of experiments

In the first set of experiments, the observations analyzed by the cdBN learnt on a given day precisely correspond to those received on the next day except that, for some of them, we add a small Gaussian noise (symbolizing that someone is tampering with the SCADA data). So `cdBN1` is used to analyze the observations of Day 2, `cdBN2` is exploited for those of `Day 3`, *etc*. For a given message,

the Gaussian noise is generated the following way: for each variable X_i in the message (except "date" and "time"), we determine the standard deviation $std(X_i)$ and the noise is equal to $0.6 \times std(X_i) \times y$, where y is a value randomly sampled from a Normal distribution $\mathcal{N}(0, 1)$. In `PL2` and `PL3`, only 57 messages and 84 messages are affected by the Gaussian noise, all the other messages of the day are kept untouched (there are about 38000 such messages per day). The modified messages are distributed evenly between Day 2 and Day 4. As an illustration, Figure 5.8 displays the values of Variable `FP_L2213` of Line `PL2`: the green dots correspond to the original values and the red ones to the perturbed values. Similarly, Figure 5.9 displays the values of Variable `P_L3_OR422` of Line `PL3`.

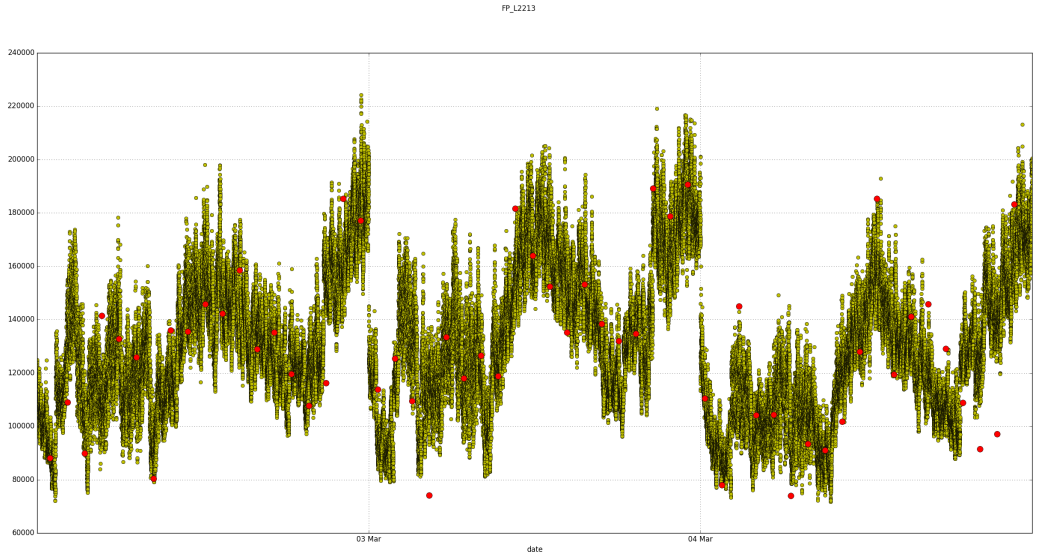


Figure 5.8: The values on Days 2 to 4 of Variable `FP_L2213` of Line `PL2`. The red dots highlight the perturbed values.

To get a clue of the overall perturbations induced by the Gaussian noise, for each message affected M, we compute two quantities μ_M and σ_M as follows: for each variable X_i (field in the message), let $\delta_i(M)$ be equal to the absolute value of the noise added to X_i divided by the absolute value of the original value of X_i. In other words, $\delta_i(M)$ represents the positive relative difference between the original value of X_i and the perturbed value. Let μ_M be the mean of the $\delta_i(M)$'s and σ_M be their standard deviation. Quantities μ_M and σ_M therefore provide information about the relative difference between the original message and the perturbed one. Figures 5.10 and 5.11 display graphically the values of μ_M and σ_M. As can be seen, the perturbations are not too high. To summarize these figures, the positive relative perturbations are equal to 0.094 ± 0.023 and 0.121 ± 0.039 for `PL2` and `PL3` respectively.

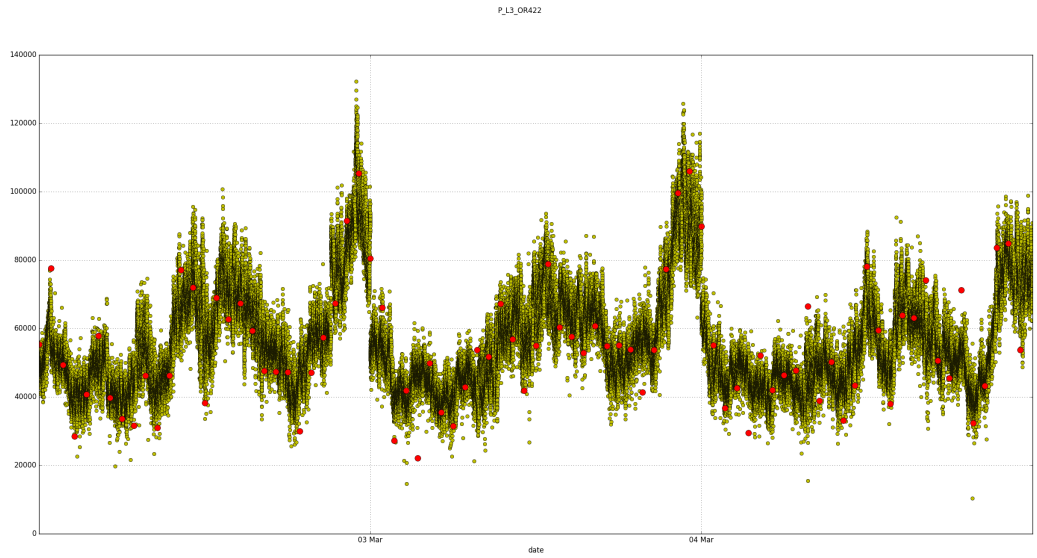


Figure 5.9: The values on Days 2 to 4 of Variable `P_L3_OR422` of Line `PL3`. The red dots highlight the perturbed values.

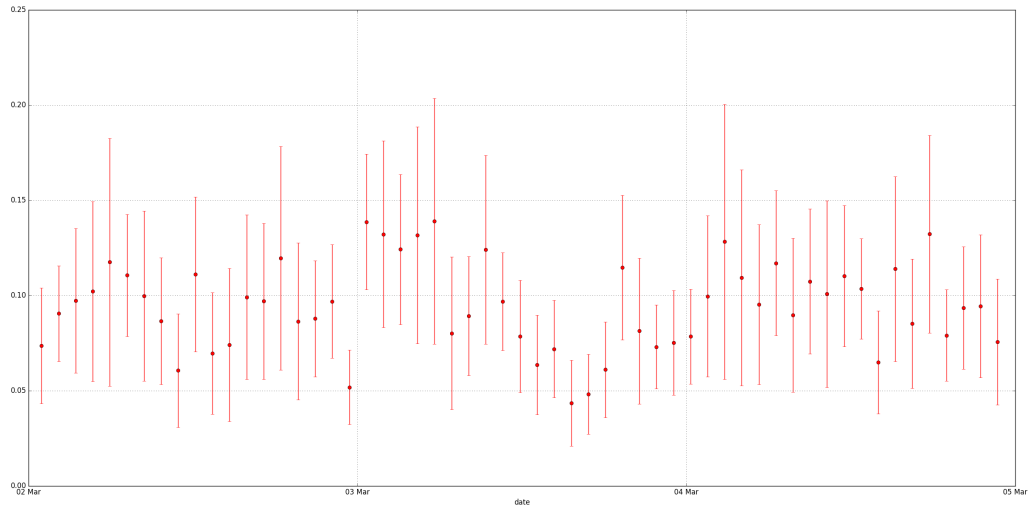


Figure 5.10: Means and standard deviations for the positive relative perturbations added to the messages in `PL2`.

Using these newly generated data, the cdBNs learnt are exploited in order to detect anomalies. Figures 5.12 and 5.13 show the results of the inferences for days 2 to 4. The blue dots represent the posteriors of the original messages whereas the red ones represent the posterior likelihoods of the perturbed messages. The horizontal red line displays the threshold and we can see that most of the red dots

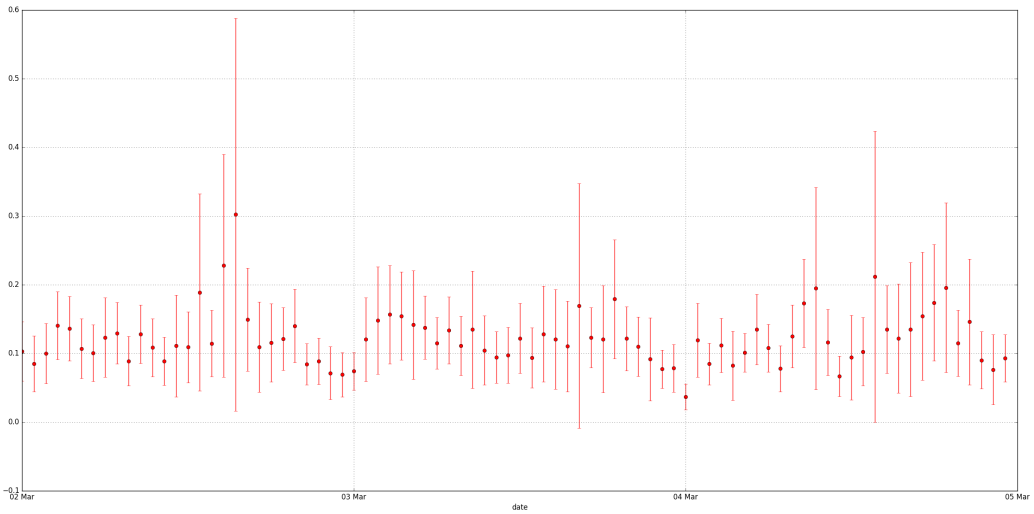


Figure 5.11: Means and standard deviations for the positive relative perturbations added to the messages in `PL3`.

are located under this line, which means that the cdBN Threat Detection Module identifies correctly the tamperings performed with the original data.

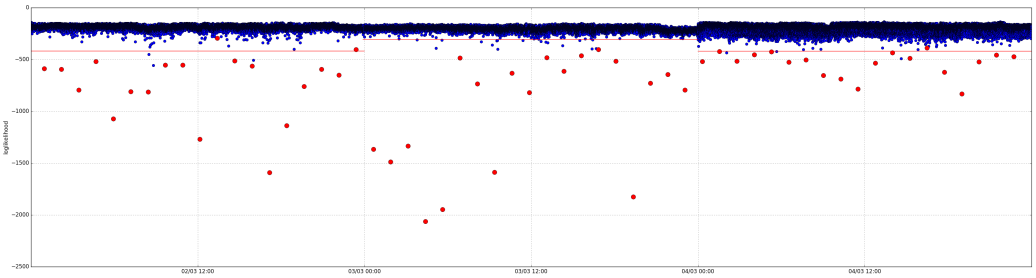


Figure 5.12: The posterior log-likelihoods of the observations from Day2 to 4 on Line `PL2`. The blue dots represent the posteriors of the original messages, the red ones represent the posterior log-likelihoods of the perturbed messages. The horizontal red line displays the log-likelihood threshold.

To capture a more high level picture of the analysis performed by the cdBN TDM, let us call false positives (hereafter denoted FP) the percentage of alarms raised by the TDM on data that were not perturbed, and true positives (hereafter denoted TP) the percentage of alarms raised on data that were perturbed. Tables 5.5 and 5.6 display for each day the FPs and TPs resulting from the TDM.

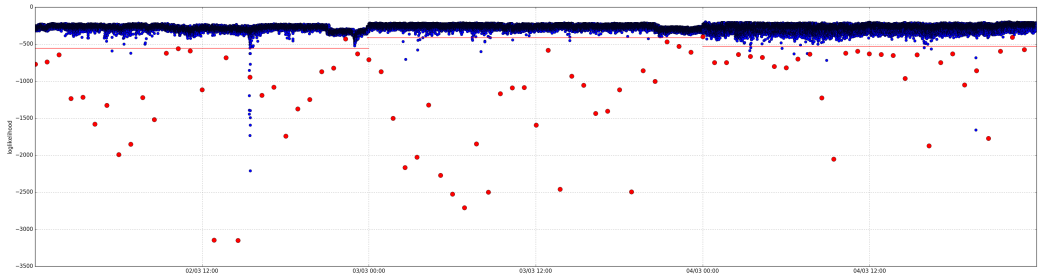


Figure 5.13: The posterior log-likelihoods of the observations from Day2 to 4 on Line PL3. The blue dots represent the posteriors of the original messages, the red ones represent the posterior log-likelihoods of the perturbed messages. The horizontal red line displays the log-likelihood threshold.

	Day 2	**Day 3**	**Day 4**	**Total**
FP	$3/37988 = 0.01\%$	$25/38418 = 0.07\%$	$3/38246 = 0.01\%$	$31/114652 = 0.03\%$
TP	$17/19 = 89.47\%$	$19/19 = 100.0\%$	$18/19 = 94.74\%$	$54/57 = 94.74\%$

Table 5.5: The TPs and FPs per day on Line PL2.

	Day 2	**Day 3**	**Day 4**	**Total**
FP	$15/37982 = 0.04\%$	$23/38425 = 0.06\%$	$10/38246 = 0.03\%$	$48/114653 = 0.04\%$
TP	$28/29 = 96.55\%$	$27/28 = 96.43\%$	$26/27 = 96.3\%$	$81/84 = 96.43\%$

Table 5.6: The TPs and FPs per day on Line PL3.

5.4.3 Second set of experiments

In the second set of experiments, we also exploit the observations of the next day but, for some of them, we substitute the values of all the variables except "date" and "time" by the values of the same variable 12 hours earlier. Here, the key idea is that the probability distributions of the data are non stationary. In particular, the distribution at time t differs from that at time $t - 12$ hours. So, by shifting the observed data by 12 hours, without shifting their time-stamp, the cdBN should identify some anomalies and those are not randomized: they are real data observed in Favignana. As for the first set of experiments, 57 messages and 84 messages randomly selected in days 2 to 4 are affected by the shifting, the other 38000 messages per day are kept untouched. As an illustration, Figure 5.14 displays the values of Variable FP_L2213 of Line PL2 (as for the preceding subsection, the green dots correspond to the original values and the red ones to the time-shifted values). Similarly, Figure 5.15 displays the values of Variable P_L3_OR422 of Line PL3.

As for the first set of experiments, in order to see the overall perturbations in-

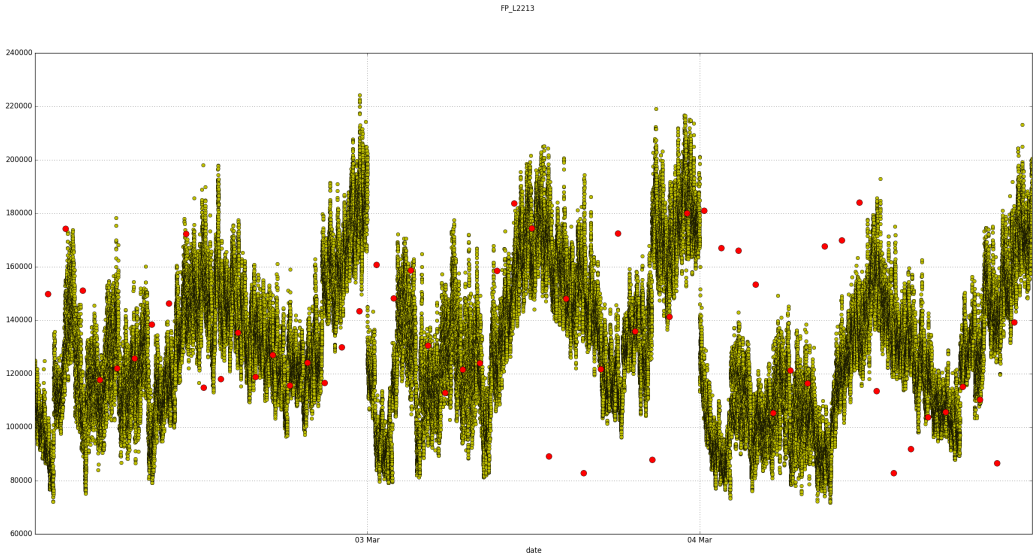


Figure 5.14: The values on Days 2 to 4 of Variable `FP_L2213` of Line `PL2`. The red dots highlight the time-shifted values.

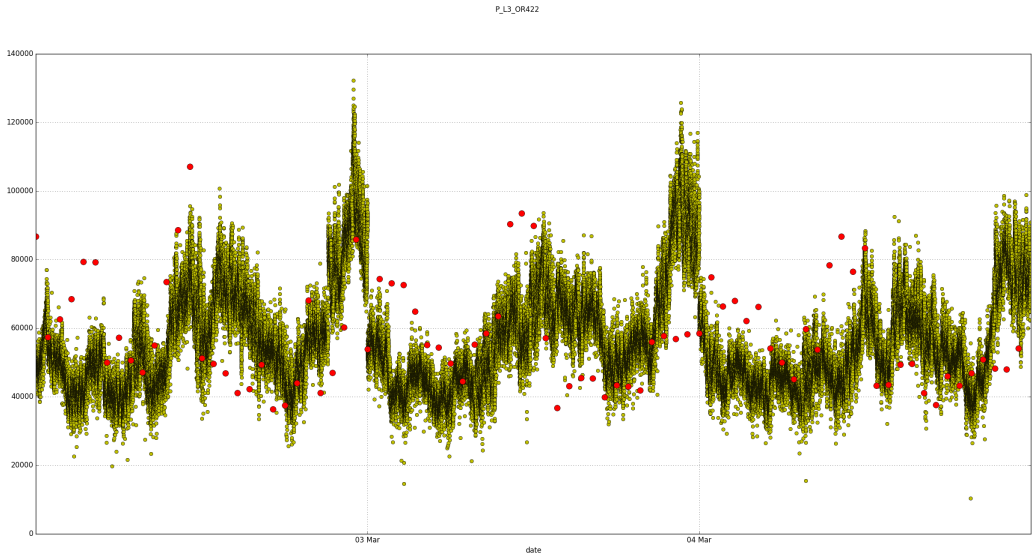


Figure 5.15: The values on Days 2 to 4 of Variable `P_L3_OR422` of Line `PL3`. The red dots highlight the time-shifted values.

duced by the time shift, for each message affected M, we compute two quantities μ_M and σ_M as follows: for each variable X_i (field in the message), let $\delta_i(M)$ be equal to the absolute value of the noise added to X_i divided by the absolute value of the original value of X_i. Let μ_M be the mean of the $\delta_i(M)$'s and σ_M be their

standard deviation. Quantities μ_M and σ_M provide information about the relative difference between the original message and the time-shifted one. Figures 5.16 and 5.17 display graphically the values of μ_M and σ_M. As can be seen, the perturbations are not too high, although they are higher than those resulting from the Gaussian noise. To summarize these figures, the positive relative perturbations are equal to 0.247 ± 0.107 and 0.287 ± 0.244 for `PL2` and `PL3` respectively.

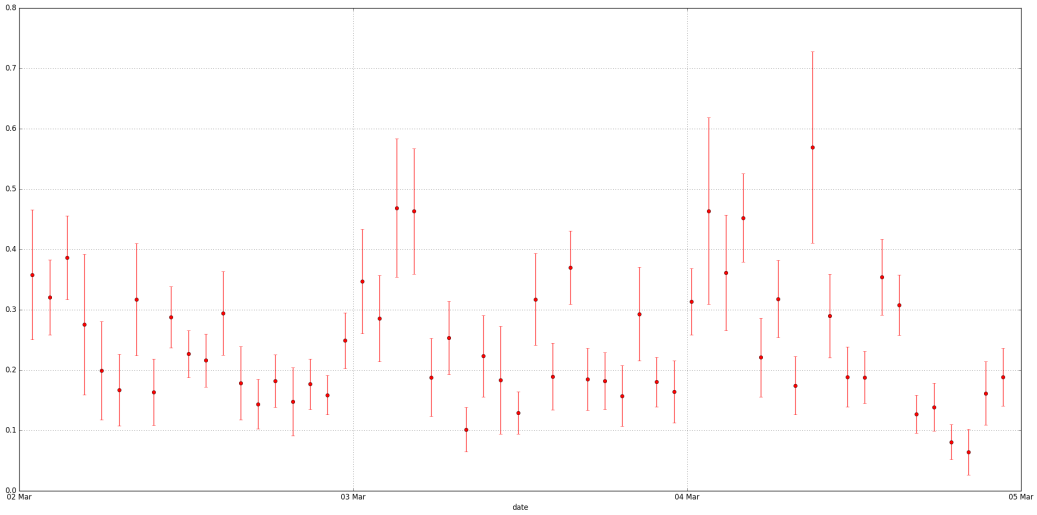


Figure 5.16: Means and standard deviations for the positive relative perturbations added to the messages in `PL2`.

Figures 5.18 and 5.19 show the results of the inferences by cdBNs of the resulting data on days 2 to 4. The blue dots represent the log-posteriors of the original messages whereas the red ones represent the posterior log-likelihoods of the perturbed messages. The horizontal red line displays the threshold and we can see that most of the red dots are located under this line, which means that the cdBN Threat Detection Module identifies correctly the tamperings performed with the original data.

To capture a more high level picture of the analysis performed by the cdBN TDM, Tables 5.7 and 5.8 display for each day the FPs and TPs resulting from the TDM.

	Day 2	**Day 3**	**Day 4**	**Total**
FP	$3/37988 = 0.01\%$	$25/38418 = 0.07\%$	$3/38246 = 0.01\%$	$31/114652 = 0.03\%$
TP	$19/19 = 100.0\%$	$19/19 = 100.0\%$	$19/19 = 100.0\%$	$57/57 = 100.0\%$

Table 5.7: The TPs and FPs per day on Line `PL2`.

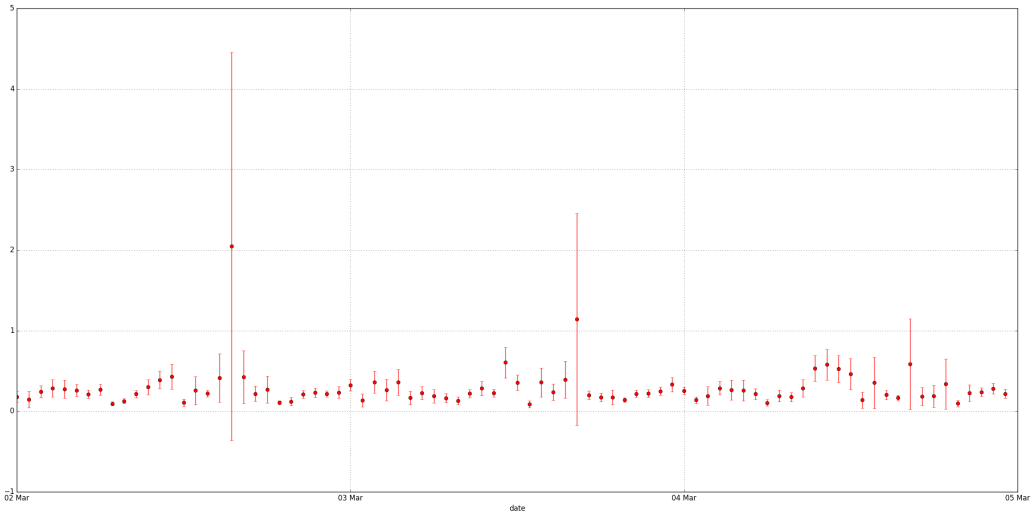


Figure 5.17: Means and standard deviations for the positive relative perturbations added to the messages in `PL3`.

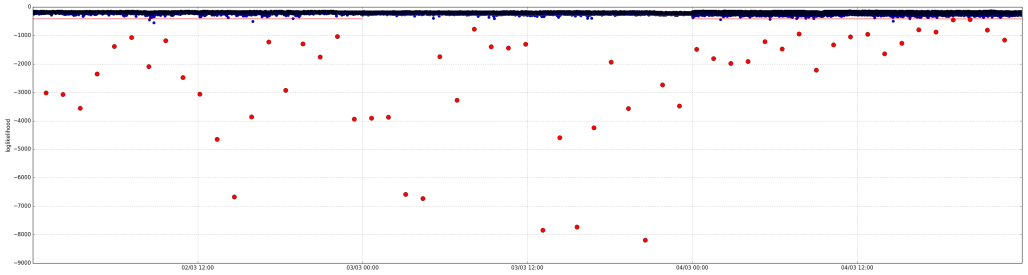


Figure 5.18: The posterior log-likelihoods of the observations from Day2 to 4 on Line `PL2`.

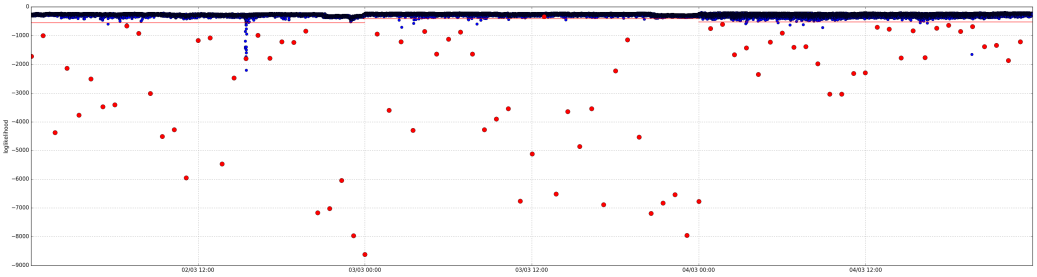


Figure 5.19: The posterior log-likelihoods of the observations from Day2 to 4 on Line `PL3`.

Overall, the above experiments show that original data tampered with either temporal noise or a small Gaussian noise are most often detected by the SCISSOR

	Day 2	**Day 3**	**Day 4**	**Total**
FP	$15/37982 = 0.04\%$	$23/38425 = 0.06\%$	$10/38246 = 0.03\%$	$48/114653 = 0.04\%$
TP	$29/29 = 100.0\%$	$27/28 = 96.43\%$	$27/27 = 100.0\%$	$83/84 = 98.81\%$

Table 5.8: The TPs and FPs per day on Line `PL3`.

cdBN Threat Detection Module. This highlights the effectiveness of the TDM, especially when observing that the tampered data are not so different from the real ones. So, semantically, for a cdBN-model, an anomaly is an observation that does not follow a given system's dynamic, which does not necessarily mean that the observation has very unlikely or aberrant values. Thanks to that, anomalies whose values are within their expected range (thus undetectable to the human eye) can be reported by the TDM module.

5.5 Conclusion

In this chapter, we have presented the general framework of the SCISSOR project and we focused on how our work has been integrated into the SCISSOR platform. This resulted in the so-called "cdBN threat detection module". In addition, we performed some experiments on real data transmitted by SEA, the Sicilian partner of the SCISSOR consortium. The domain experts suggested to exploit Kalman filters to detect which data corresponded to threats. We therefore compared the cdBNs presented in the preceding chapter with Kalman filters. This enabled to highlight the superiority of cdBNs over Kalman filters. In our point of view, this results from two features that our models possess, unlike Kalman filters: first, they are well suited to model very complex, nonlinear, non-Gaussian situations; and ii) the ns-cdBN are capable of taking into account non-stationarity. These features are important for the model to predict accurately what happens in the SCADA system. In a second set of experiments, we tampered with real data in order to simulate attackers that gradually change the system, with the ultimate purpose to make it fail. We showed that cdBNs are capable of identifying such tamperings way before those can be detected by the human operator, i.e., when looking at the tampered data, they do not seem to be erroneous, yet our probabilistic model can detect that they deviate already too much w.r.t. what was expected.

Conclusions and Future Research

Conclusions

Detecting cyber-security threats in SCADA systems is a challenging task. Its main difficulty is that such events are rare and are not expected to be similar one to another. To cope with this problem, in the SCISSOR project, many testbeds have been implemented in order to test different systems and different situations. In the context of this work, corresponding to the Decision & Analysis Layer of the mentioned project, the main challenge has been to learn BN-based models from the dynamics of a system as complex as SCADA.

Since most of the BN works in the literature are about discrete BNs, in Chapter 1, we gave a non-exhaustive introduction to the foundations of BNs, their concepts, and some of their properties. We studied two of their main inference algorithms: Variable Elimination and Shafer-Shenoy (the computations they perform being equivalent in terms of time complexity). We presented as well some techniques to learn BNs from datasets, with a special focus on score based methods. Then, in this chapter we described the concept of dynamic BNs (DBNs), which are BNs representing stationary temporal processes, i.e., the dynamics of such processes remains the same over time. Finally, we defined non-stationary DBNs, which are an extension of DBNs enabling to bypass this restriction.

Unfortunately, SCADA's random variables are most of the time of a continuous nature. This is the reason why, in Chapter 2, we mentioned the state-of-the-art alternatives to deal with this type of variables. The simplest one consists of simply discretizing all the continuous variables. Finding the right discretization in a BN context is not an easy task since we are generally not allowed to use many discretization intervals: this is essentially due to the fact that both BN inference and structure learning are NP-Hard problems, which makes all the algorithms intractable when the domain sizes of the variables are too high. As a consequence, most typical discretization methods would loose a large amount of useful information, thereby reducing the prediction/diagnosis power of the BNs.

This is especially true when discretizing independently each variable with equidistant/equiprobable intervals. In Chapter 2, we focused on discretization methods well adapted to BN's features (i.e. discretizations that take into account the BN structure) like [Monti and Cooper, 1998] and [Friedman and Goldszmidt, 1996]. An alternative to discretization consists in directly handling continuous variables. We detailed in Chapter 2 two of the main state-of-the-art BN-based models of this family: CLGs and MTBFs. The former make the assumption that every continuous variable is a linear combination of its continuous parents in the BN structure (for a fixed value of its discrete parents) and that it follows a conditional Gaussian distribution. In addition, no discrete random variable is allowed to have a continuous variable as parent. This leads us to the interesting property that, for any valid instantiation of its discrete variables, the resulting distribution of a CLG is a multivariate normal distribution. As a consequence, after any variable's marginalization, the resulting distribution remains a multivariate normal distribution. This allows for very fast inference algorithms. We also briefly studied MTBFs models. They rely on basis functions like exponentials, polynomials, beta functions, etc., to represent different-shaped multivariate density functions as mixtures of simpler distributions. Although they might struggle representing conditional density functions, they give a very accurate joint distribution representation, but this come with a cost: their inference times are in general prohibitive because the products of their basis functions are prone to generate large algebraic expressions.

In chapter 3, we presented our first contribution: a BN-based model which makes a trade-off between the efficiency of inference in CLGs and the expressive power of MTBFs. We called this model the *Conditional Truncated Densities Bayesian Network* (ctdBN). A ctdBN is an hybrid BN model[11]. In this model, each continuous variable $\mathring{X}_i$ has a *discretized* counterpart X_i. The BN structure is such that X_i is the only parent of $\mathring{X}_i$. We use CPTs for the discrete variable relationships (just like in a discrete BN), and we define a conditional truncated density function to describe the density $f(\mathring{X}_i = \mathring{x}_i | X_i = x_i)$. This function is meant to be truncated within the discretization interval of $\mathring{X}_i$ corresponding to Value x_i of X_i. In Chapter 3, we showed that ctdBNs are capable to approach any Lipschitz multivariate mixed probability distribution (which includes almost all the most commonly used probability density functions). We also showed that the inference complexity in ctdBNs remains of the same order as one in a discrete BN.

In chapter 4, we described a critical issue that the ctdBN model might have when being learnt from a dataset using a score-based approach: it might produce undesirable (too compact) discretization intervals which, in practice, would not re-

[11] Hybrid = it can handle both discrete and continuous random variables at the same time

flect the dataset's variables distribution. To overcome this problem, we proposed a (more general) model of hybrid BNs, which we called *conditional densities Bayesian networks* (cdBN). This model keeps the key ideas of the ctdBNs: for each continuous variable $\mathring{X}_i$, there must be a corresponding discrete variable X_i. But this discrete variable is not a discretization of its counterpart anymore because we model the function $f(\mathring{X}_i = \mathring{x}_i | X_i = x_i)$ as an untruncated density function rather than as a truncated one. In this context, Variable X_i should be interpreted as a hidden variable (when learning) more than as a discretized one. The cdBN model also compels the existence of the edge $X_i \to \mathring{X}_i$ in the BN structure, but it allows for the existence of the arcs of the form $X_j \to \mathring{X}_i$ whenever $X_j \in \mathbf{Pa}(X_i)$, making it an even more expressive model. We showed that, although the continuous variables can have these extra parents, the inference complexity remains of the same order as inference in the BN discrete part of the cdBN. Moreover, we showed that the cdBN model is at least as expressive as the ctdBN one (which makes sense, because we can consider ctdBNs as a particular case of cdBNs). We derived a score function in order to learn the cdBN model from a database as well as a learning algorithm based on Structural EM [Friedman, 1998], considering (as previously mentioned) the "discretized" variables in the cdBN as hidden.

Finally, in Chapter 5 we briefly described the general architecture of the SCISSOR framework, as well as our contributions in this project. We used the cdBN model as an anomaly detection tool, applying it to the values sent by the sensors of the power lines of the Favignana's electric power plant. The approach we used for detecting anomalies is simple: we have a virtual machine containing the current cdBN model of each power line (those are all independent according to Favignana's experts), and this virtual machine receives values of the current state of the system in real time. Having defined a likelihood threshold, we raise an alert to the central SCADA system whenever the current state of any power line is unlikely, i.e., below the threshold. The current cdBN model is learnt in batch mode from data sent in the preceding hours. For each continuous variable $\mathring{X}$, we also considered a *differential* variable $\mathring{X}'$ (which in fact makes the cdBN model a non-stationary and temporal model). The cdBN model and all the routines it uses were programmed using the aGrUM library[12]. The goal of the SCISSOR project is to detect anomalies that could be caused by cyber-attacks. Since those are very seldom encountered, it was not possible to get real Favignana data in which such attacks occurred. As a consequence, for experiements, we simulated such attacks by tampering progressively with the "true" data sent by sensors through the Favignana network. This was to be interpreted as an attacker modifying slightly the system in order to produce electricity failures, notably by inverting the power

[12] `agrum.gitlab.io/`

flux within the electrical network. Therefore, we limited our experiments to data perturbations generated by ourselves following guidelines given by Favignana's experts. We performed comparisons between our model and the Kalman filter (which is a commonly used model for this task). And we integrated our model into the testbed protocol, successfully validating its good properties for detecting anomalies.

Future Research

The BN-based models designed in this thesis are well suited for detecting anomalies. Yet, much efforts need to be done in order to make them as accessible as discrete BNs. In the following paragraphs we mention some perspectives to this thesis.

CdBNs (as well as ctdBNs) are agnostic of the shape of the conditional density functions $f(\mathring{X}|\mathbf{Y})$. However, for the moment, we have explored exclusively the use of mixtures of normal distributions (sometimes truncated) to describe them. An interesting perspective to this research (inspired by the basis functions of MTBFs) would be to explore the behavior of such models when other types of density functions are used, and derive the cdBN learning score for each one of those different types of conditional densities $f(\cdot)$.

The current cdBN learning score makes the assumption that the domain sizes of the hidden ("discretized") variables are already given. In the experiments of the present work, we obtained these values by performing a discretization algorithm like [Friedman and Goldszmidt, 1996], or fixing them manually. In this context, finding the "correct" domain size of the hidden counterpart of a continuous variable is equivalent to the unsupervised learning problem of deciding the right amount of clusters it might have in a given dataset. This suggests to borrow some criteria from the Machine learning community like the *elbow criterion*, or performing a k-means clusterization with an arbitrarily big number of clusters and reducing that dimension whenever we found empty clusters. We could also directly use a BIC or AIC criterion to penalize the number of clusters, or to adapt the method presented in [Elidan and Friedman, 2001] to the models presented in this thesis.

The cdBN and ctdBN models have exact-inference times of the same complexity as those of discrete BNs. However, the current cdBN learning algorithm is based on the Structural EM algorithm (SEM), which considers the discrete coun-

terparts of the continuous variables as hidden. This makes the learning process considerably slow: all the possible hidden variables' instantiations must be considered when evaluating the score value for each iteration (which increases exponentially in the number of hidden variables). There is work to be done on how to optimize the SEM convergence time. One clue into this direction is to use a discretization algorithm to find an initial BN structure and to start the SEM search from this structure. Another clue is to use a *k-means approach* rather than SEM in a first step: using k-means, we can substitute the unobserved values in the dataset by only one value (the one with the highest probability). Once this is done, the search in the DAG space can be performed as usual on a complete dataset (without needing performing many cdBN inferences as SEM would require). Upon convergence, we can then switch to an SEM-based searching using the resulting structure as a starting point. Since the SEM algorithm contains whole EM processes at each iteration, another interesting clue to accelerate it is by applying any of the many state-of-the-art methods to accelerate EM [Shioya and Miura, 2011, Varadhan and Roland, 2008, Guo et al., 2017, Ortiz and Kaelbling, 1999].

When using Gaussian conditional densities, there is a serious problem when performing EM iterations: this is caused by potential singularities because the likelihood of a Gaussian mixtures is not necessarily bounded: there could be Gaussian functions with close-to-zero-value variance (which might look like a Dirac delta function). Fortunately, in [Redner and Walker, 1984] it is proved that, although we could fall into singularities, there will always exist consistent EM solutions, so it would be enough to relaunch the algorithm an indefinite number of times (with different initial values) to achieve one of those. However, reinitializing the EM algorithm might be tedious because the learning algorithm is very time consuming. There is research work to be done about better ways to handle this type of situations. One possible solution (vaguely explored in this thesis) is to assume *a priori* that each continuous variable of the learning dataset follows a Gaussian distribution with a given variance. There remains work to be done to justify this assumption and, moreover, to develop a method to assign a pertinent value to that variance.

Although for the SCISSOR project we used (in practice) non-stationary dynamic cdBN models, those were not learnt as such: The learning process was performed in batches just as if they were regular cdBNs; the epoch changing times were chosen by hand (helped by knowledge of experts); there was no notion of transition from one model to another across epochs. There is much research to be done on how to perform these tasks more properly. Moreover, our study case was a SCADA system in which new variables could be added while others could be temporary/permanently removed from the system. There is a lot of research work

to be done on how to deal with such cases.

In applications like SCADA, variables are not always labeled as discrete or continuous ones. And even when they are labeled, there are cases when we do not know *a priori* all their possible values. Some continuous variables have a "discrete" or "semi-discrete" behavior. For instance, in the SCISSOR project, we dealt with many variables such as the one plotted in Figure 5.7 (Page 163): Variable `Q_L2_STIMATA1` (the third one), is identified as a continuous variable, but it actually behaves mostly like a discrete one; in the same figure, the values of Variable `FQ_L2211` (the second one) remain almost fixed for a long period of time, and in other periods they are clearly of a continuous nature. These types of situation were really problematic in the present work approach: note that when approximating those variables by a mixture of normal distributions, we are necessarily going to fall into a singularity. How to automatically identify those types of variables and how to model them in the ctdBN/cdBN context remains an open question.

There are ways to learn missing or hidden discrete variables in the BN context [Elidan, 2004], that can be adapted into our models. However, the impact of hidden continuous variables in cdBNs remains an open research subject. Moreover, there is work to be done looking for introducing other learning approaches for the models presented in this thesis, like adapting constraint based methods into it (note that independences between a continuous and a discrete variable could be asserted using ANOVA tests [Gelman, 2010]).

Bibliography

[Abellán et al., 2006] Abellán, J., Gómez-Olmedo, M., Moral, S., et al. (2006). Some variations on the pc algorithm. In *Probabilistic Graphical Models*, pages 1–8.

[Acid and De Campos, 1996] Acid, S. and De Campos, L. M. (1996). An algorithm for finding minimum d-separating sets in belief networks. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pages 3–10. Morgan Kaufmann Publishers Inc.

[Akaike, 1970] Akaike, H. (1970). Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22(1):203–217.

[Andreassen et al., 1992] Andreassen, S., Falck, B., and Olesen, K. G. (1992). Diagnostic function of the microhuman prototype of the expert system — munin. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 85(2):143 – 157.

[Arnborg et al., 1987] Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284.

[Baron, 1987] Baron, J. (1987). Second-order probabilities and belief functions. *Theory and Decision*, 23(1):25–36.

[Bottone et al., 2008] Bottone, S., Lee, D., O'Sullivan, M., and Spivack, M. (2008). Failure prediction and diagnosis for satellite monitoring systems using bayesian networks. pages 1 – 7.

[Buntine, 1991] Buntine, W. (1991). Theory refinement on bayesian networks. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pages 52–60. Morgan Kaufmann Publishers Inc.

[Butz et al., 2010] Butz, C., Yan, W., Lingras, P., and Yao, Y. (2010). The CPT structure of variable elimination in discrete Bayesian networks. In Ras, Z. and

Tsay, L., editors, *Advances in Intelligent Information Systems*, pages 245–257. Springer.

[Campos and Ji, 2011] Campos, C. P. d. and Ji, Q. (2011). Efficient structure learning of bayesian networks using constraints. *Journal of Machine Learning Research*, 12(Mar):663–689.

[Campos, 2006] Campos, L. M. d. (2006). A scoring function for learning bayesian networks based on mutual information and conditional independence tests. *Journal of Machine Learning Research*, 7(Oct):2149–2187.

[Chavira and Darwiche, 2006] Chavira, M. and Darwiche, A. (2006). Encoding cnfs to empower component analysis. In Biere, A. and Gomes, C. P., editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, pages 61–74, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Chavira and Darwiche, 2008] Chavira, M. and Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772 – 799.

[Cheng and Greiner, 2013] Cheng, J. and Greiner, R. (2013). Comparing bayesian network classifiers. *CoRR*, abs/1301.6684.

[Cheng et al., 2002] Cheng, J., Greiner, R., Kelly, J., Bell, D., and Liu, W. (2002). Learning bayesian networks from data: an information-theory based approach. *Artificial intelligence*, 137(1-2):43–90.

[Cheng et al., 2013] Cheng, Y., Xu, T., and Yang, L. (2013). Bayesian network based fault diagnosis and maintenance for high-speed train control systems. *2013 International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE)*, pages 1753–1757.

[Chickering, 1996] Chickering, D. (1996). Learning bayesian networks is np-complete. In Fisher, D. and Lenz, H.-J., editors, *Learning from Data*, volume 112 of *Lecture Notes in Statistics*, pages 121–130. Springer New York.

[Chickering, 1995] Chickering, D. M. (1995). A transformational characterization of equivalent bayesian network structures. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 87–98. Morgan Kaufmann Publishers Inc.

[Chickering, 2002a] Chickering, D. M. (2002a). Learning equivalence classes of bayesian-network structures. *Journal of machine learning research*, 2(Feb):445–498.

[Chickering, 2002b] Chickering, D. M. (2002b). Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554.

[Chickering and Meek, 2006] Chickering, D. M. and Meek, C. (2006). On the incompatibility of faithfulness and monotone dag faithfulness. *Artificial Intelligence*, 170(8-9):653–666.

[Chow and Liu, 1968] Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467.

[Cobb and Shenoy, 2006] Cobb, B. and Shenoy, P. (2006). Inference in hybrid Bayesian networks with mixtures of truncated exponentials. *International Journal of Approximate Reasoning*, 41(3):257–286.

[Cobb et al., 2006] Cobb, B., Shenoy, P., and Rumí, R. (2006). Approximating probability density functions in hybrid Bayesian networks with mixtures of truncated exponentials. *Statistics and Computing*, 16(3):293–308.

[Colombo and Maathuis, 2014] Colombo, D. and Maathuis, M. H. (2014). Order-independent constraint-based causal structure learning. *The Journal of Machine Learning Research*, 15(1):3741–3782.

[Cooper, 1990] Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393 – 405.

[Cooper and Herskovits, 1992a] Cooper, G. F. and Herskovits, E. (1992a). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347.

[Cooper and Herskovits, 1992b] Cooper, G. F. and Herskovits, E. (1992b). A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[Cortijo and Gonzales, 2016] Cortijo, S. and Gonzales, C. (2016). Bayesian networks with conditional truncated densities. In *Proc. of the Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 656–661.

[Cover and Thomas, 2006] Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA.

[Cussens, 2012] Cussens, J. (2012). Bayesian network learning with cutting planes. *arXiv preprint arXiv:1202.3713*.

[Cussens et al., 2017] Cussens, J., Järvisalo, M., Korhonen, J. H., and Bartlett, M. (2017). Bayesian network structure learning with integer programming: Polytopes, facets and complexity. *Journal of Artificial Intelligence Research*, 58:185–229.

[Dagum and Luby, 1993] Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141 – 153.

[Dasgupta, 1999] Dasgupta, S. (1999). Learning polytrees. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 134–141. Morgan Kaufmann Publishers Inc.

[Dash and Druzdzel, 2012] Dash, D. and Druzdzel, M. J. (2012). A robust independence test for constraint-based learning of causal structure. *CoRR*, abs/1212.2464.

[De Campos et al., 2009] De Campos, C. P., Zeng, Z., and Ji, Q. (2009). Structure learning of bayesian networks using constraints. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 113–120. ACM.

[De Campos et al., 2002] De Campos, L. M., Fernández-Luna, J. M., and Puerta, J. M. (2002). Local search methods for learning bayesian networks using a modified neighborhood in the space of dags. In *Ibero-American Conference on Artificial Intelligence*, pages 182–192. Springer.

[Dean and Kanazawa, 1989] Dean, T. and Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150.

[Debar et al., 2007] Debar, H., Curry, D., and Feinstein, B. (2007). The intrusion detection message exchange format (idmef). `http://tools.ietf.org/rfc/rfc4765.txt`.

[Dechter, 1999] Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85.

[Dondelinger et al., 2010] Dondelinger, F., Lèbre, S., and Husmeier, D. (2010). Heterogeneous continuous dynamic Bayesian networks with flexible structure and inter-time segment information sharing. In *Proceedings of the International Conference on Machine Learning*, pages 303–310.

[Elidan, 2004] Elidan, G. (2004). *Learning Hidden Variables in Probabilistic Graphical Models*. PhD thesis, Hebrew University of Jerusalem.

[Elidan and Friedman, 2001] Elidan, G. and Friedman, N. (2001). Learning the dimensionality of hidden variables. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 144–151. Morgan Kaufmann Publishers Inc.

[Elidan et al., 2002] Elidan, G., Ninio, M., Friedman, N., and Shuurmans, D. (2002). Data perturbation for escaping local maxima in learning. In *AAAI/IAAI*, pages 132–139.

[Fast et al., 2008] Fast, A., Hay, M., and Jensen, D. (2008). Improving accuracy of constraint-based structure learning. Technical report, Technical report 08-48, University of Massachusetts Amherst, Computer Science Department.

[Fenton and Neil, 2012] Fenton, N. and Neil, M. (2012). *Risk Assessment and Decision Analysis with Bayesian Networks*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

[Friedman, 1997] Friedman, Geiger, G. (1997). Bayesian network classifiers. *Machine Learning*, pages 131–163.

[Friedman, 1998] Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proc. of Uncertainty in Artificial Intelligence*, pages 129–138.

[Friedman and Goldszmidt, 1996] Friedman, N. and Goldszmidt, M. (1996). Discretizing continuous attributes while learning Bayesian networks. In *proceedings of ICML'96*, pages 157–165.

[Friedman et al., 2000] Friedman, N., Linial, M., Nachman, I., and Pe'er, D. (2000). Using bayesian networks to analyze expression data. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00, pages 127–135, New York, NY, USA. ACM.

[Friedman et al., 1999] Friedman, N., Nachman, I., and Peér, D. (1999). 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.

[Galland, 2010] Galland, J.-P. (2010). Critique de la notion d'infrastructure critique. *Flux*, 81(3):6–18.

[Gámez et al., 2011] Gámez, J. A., Mateo, J. L., and Puerta, J. M. (2011). Learning bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery*, 22(1-2):106–148.

[Geiger and Heckerman, 1995] Geiger, D. and Heckerman, D. (1995). A characterization of the Dirichlet distribution with application to learning Bayesian networks. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, pages 196–207.

[Geiger and Pearl, 1990] Geiger, D. and Pearl, J. (1990). On the logic of causal models. In *Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '88, pages 3–14, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

[Gelman, 2010] Gelman, A. (2010). Variance, Analysis of. In Durlauf, S. N. and Blume, L. E., editors, *Microeconometrics*, pages 339–347. Palgrave Macmillan UK, London.

[Geman and Geman, 1984] Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

[Gillispie and Perlman, 2001] Gillispie, S. B. and Perlman, M. D. (2001). Enumerating markov equivalence classes of acyclic digraph dels. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 171–177. Morgan Kaufmann Publishers Inc.

[Glover, 1990] Glover, F. (1990). Tabu search—part ii. *ORSA Journal on computing*, 2(1):4–32.

[Gonzales, 2008] Gonzales, C. (2008). Inférence dans les réseaux bayésiens. In *Cours de MGDE*. Master Informatique, Université Pierre et Marie Curie.

[Gonzales et al., 2015] Gonzales, C., Dubuisson, S., and Manfredotti, C. (2015). A new algorithm for learning non-stationary dynamic Bayesian networks with application to event detection. In *Proc. of the Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 564–569.

[Grünwald, 2007] Grünwald, P. D. (2007). *The minimum description length principle*. MIT press.

[Grzegorczyk and Husmeier, 2009] Grzegorczyk, M. and Husmeier, D. (2009). Non-stationary continuous dynamic Bayesian networks. In *Proc. of the International Conference Advances in Neural Information Processing Systems (NIPS)*, pages 682–690.

[Guo et al., 2017] Guo, X., Li, Q.-y., and Xu, W.-l. (2017). Acceleration of the em algorithm using the vector aitken method and its steffensen form. *Acta Mathematicae Applicatae Sinica, English Series*, 33(1):175–182.

[Hastings, 1970] Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

[Heckerman, 1995] Heckerman, D. (1995). A tutorial on learning with Bayesian networks. Technical report, Microsoft Research.

[Heckerman and Geiger, 1995] Heckerman, D. and Geiger, D. (1995). Learning bayesian networks: a unification for discrete and gaussian domains. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 274–284. Morgan Kaufmann Publishers Inc.

[Heckerman et al., 1995] Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243.

[Heckerman et al., 1992] Heckerman, D., Horvitz, E., Laboratory, S. U. C. S. D. K. S., and Nathwani, B. (1992). *Toward Normative Expert Systems: Part I, the Pathfinder Project*. Number pt. 1 in Report (Stanford University. Knowledge Systems Laboratory). Knowledge Systems Laboratory, Medical Computer Science, Stanford University.

[Henrion, 1988] Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Machine Intelligence and Pattern Recognition*, 5:149–163.

[Huang et al., 2014] Huang, Y., Wang, Y., and Zhang, R. (2014). Fault troubleshooting using bayesian network and multicriteria decision analysis. *Advances in Mechanical Engineering*, 6.

[Jaakkola et al., 2010] Jaakkola, T., Sontag, D., Globerson, A., and Meila, M. (2010). Learning bayesian network structure using lp relaxations. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 358–365.

[Jeffreys, 1946] Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proc. R. Soc. Lond. A*, 186(1007):453–461.

[Karnouskos, 2011] Karnouskos, S. (2011). Stuxnet worm impact on industrial cyber-physical system security. In *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*, pages 4490–4494. IEEE.

[Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

[Kindermann et al., 1980] Kindermann, R., Snell, J., and Society, A. M. (1980). *Markov Random Fields and Their Applications*. AMS books online. American Mathematical Society.

[Kjærulff, 1990] Kjærulff, U. (1990). Triangulation of graphs – algorithms giving small total state space. Technical report.

[Koivisto and Sood, 2004] Koivisto, M. and Sood, K. (2004). Exact bayesian structure discovery in bayesian networks. *Journal of Machine Learning Research*, 5(May):549–573.

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[Lam and Bacchus, 1994] Lam, W. and Bacchus, F. (1994). Learning bayesian belief networks: An approach based on the mdl principle. *Computational intelligence*, 10(3):269–293.

[Langseth et al., 2012a] Langseth, H., Nielsen, T., Rumí, R., and Salmerón, A. (2012a). Inference in hybrid Bayesian networks with mixtures of truncated basis functions. In *Proceedings of PGM'12*, pages 171–178.

[Langseth et al., 2012b] Langseth, H., Nielsen, T., Rumí, R., and Salmerón, A. (2012b). Mixtures of truncated basis functions. *International Journal of Approximate Reasoning*, 53(2):212–227.

[Lauritzen, 1992] Lauritzen, S. (1992). Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87:1098–1108.

[Lauritzen and Wermuth, 1989] Lauritzen, S. and Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17(1):31–57.

[Lauritzen and Jensen, 2001] Lauritzen, S. L. and Jensen, F. (2001). Stable local computation with conditional gaussian distributions. *Statistics and Computing*, 11(2):191–203.

[Lebeltel et al., 2004] Lebeltel, O., Bessière, P., Diard, J., and Mazer, E. (2004). Bayesian robot programming. *Autonomous Robots*, 16(1):49–79.

[Lee et al., 2016] Lee, R. M., Assante, M. J., and Conway, T. (2016). Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*.

[Lichman, 2013] Lichman, M. (2013). UCI machine learning repository.

[Liu et al., 2017] Liu, H., Zhou, S., Lam, W., and Guan, J. (2017). A new hybrid method for learning bayesian networks: Separation and reunion. *Knowledge-Based Systems*, 121:185–197.

[Mabrouk et al., 2015] Mabrouk, A., Gonzales, C., Jabet-Chevalier, K., and Chojnaki, E. (2015). Multivariate cluster-based discretization for Bayesian network structure learning. In *Proceedings of SUM'15*.

[Madsen and Jensen, 1999] Madsen, A. and Jensen, F. (1999). LAZY propagation: A junction tree inference algorithm based on lazy inference. *Artificial Intelligence*, 113(1–2):203–245.

[Mateescu et al., 2010] Mateescu, R., Kask, K., Gogate, V., and Dechter, R. (2010). Join-graph propagation algorithms. *Journal of Artificial Intelligence Research*, 37:279–328.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Meek, 1995a] Meek, C. (1995a). Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 403–410. Morgan Kaufmann Publishers Inc.

[Meek, 1995b] Meek, C. (1995b). Strong completeness and faithfulness in bayesian networks. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 411–418. Morgan Kaufmann Publishers Inc.

[Monti and Cooper, 1998] Monti, S. and Cooper, G. (1998). A multivariate discretization method for learning bayesian networks from mixed data. In *Proc. of UAI'98*, pages 404–413.

[Moore and Wong, 2003] Moore, A. and Wong, W.-K. (2003). Optimal reinsertion: A new search operator for accelerated and more accurate bayesian network structure learning. In *ICML*, volume 3, pages 552–559.

[Moral et al., 2001] Moral, S., Rumí, R., and Salmerón, A. (2001). Mixtures of truncated exponentials in hybrid Bayesian networks. In *Proceedings of ECSQARU'01*, volume 2143 of *LNAI*, pages 156–167.

[Moral et al., 2002] Moral, S., Rumí, R., and Salmerón, A. (2002). Estimating mixtures of truncated exponentials from data. In *Proceedings of PGM'02*, pages 135–143.

[Murphy and Mian, 1999] Murphy, K. and Mian, S. (1999). Modelling gene expression data using dynamic bayesian networks.

[Nielsen and Nielsen, 2008] Nielsen, S. H. and Nielsen, T. D. (2008). Adapting Bayes nets to non-stationary probability distributions. *International Journal of Approximate Reasoning*, 49(2):379–397.

[Ortiz and Kaelbling, 1999] Ortiz, L. E. and Kaelbling, L. P. (1999). Accelerating em: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 512–521. Morgan Kaufmann Publishers Inc.

[Ott et al., 2003] Ott, S., Imoto, S., and Miyano, S. (2003). Finding optimal models for small gene networks. In *Biocomputing 2004*, pages 557–567. World Scientific.

[Pantel and Lin, 1998] Pantel, P. and Lin, D. (1998). Spamcop: A spam classification & organization program. In *In Learning for Text Categorization: Papers from the 1998 Workshop*, pages 95–98.

[Park and Darwiche, 2003] Park, J. and Darwiche, A. (2003). Solving MAP exactly using systematic search. In *Proc. of Uncertainty in Artificial Intelligence (UAI)*, pages 459–468.

[Park and Darwiche, 2004] Park, J. and Darwiche, A. (2004). Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, 21(1):101–103.

[Peña et al., 2000] Peña, J., Lozano, J., and Larrañaga, P. (2000). An improved Bayesian structural EM algorithm for learning Bayesian networks for clustering. *Pattern Recognition Letters*, 21(8):779–786.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman.

[Pearl, 2000] Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge University Press, Cambridge, England.

[Pearl, 2009] Pearl, J. (2009). *Causality*. Cambridge university press.

[Pearl and Paz, 1986] Pearl, J. and Paz, A. (1986). Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z? In *ECAI*.

[Pearl and Verma, 1991] Pearl, J. and Verma, T. (1991). A theory of inferred causation. In *Proceedings of Principles of Knowledge Representation and Reasoning (KR),*.

[Redner and Walker, 1984] Redner, R. A. and Walker, H. F. (1984). Mixture densities, maximum likelihood and the em algorithm. *SIAM review*, 26(2):195–239.

[Robinson and Hartemink, 2008] Robinson, J. W. and Hartemink, A. J. (2008). Non-stationary dynamic Bayesian networks. In *Proceedings of the Conference Advances in Neural Information Processing Systems*, pages 1369–1376.

[Robinson and Hartemink, 2010] Robinson, J. W. and Hartemink, A. J. (2010). Learning non-stationary dynamic Bayesian networks. *Journal of Machine Learning Research*, 11:3647–3680.

[Romero et al., 2004] Romero, R., Rumí, R., and Salmerón, A. (2004). Structural learning of Bayesian networks with mixtures of truncated exponentials. In *Proceedings of PGM'04*, pages 177–184.

[Rose, 1970] Rose, D. J. (1970). Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597 – 609.

[Rumíand Salmerón, 2007] Rumí, R. and Salmerón, A. (2007). Approximate probability propagation with mixtures of truncated exponentials. *International Journal of Approximate Reasoning*, 45(2):191–210.

[Sahami et al., 1998] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998). A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*. AAAI Technical Report WS-98-05.

[Schlaifer and Raiffa, 1961] Schlaifer, R. and Raiffa, H. (1961). *Applied statistical decision theory*.

[Schulte et al., 2009] Schulte, O., Frigo, G., Greiner, R., Luo, W., and Khosravi, H. (2009). A new hybrid method for bayesian network learning with dependency constraints. In *Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on*, pages 53–60. IEEE.

[Schumann et al., 2012] Schumann, J., Mbaya, T., Mengshoel, O. J., and Schumann, J. M. (2012). Software and system health management for autonomous robotics missions.

[Schwarz, 1978] Schwarz, G. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

[Scott, 1992] Scott, D. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, New York.

[Scutari, 2016] Scutari, M. (2016). An empirical-bayes score for discrete bayesian networks. In *Conference on Probabilistic Graphical Models*, pages 438–448.

[Scutari, 2017] Scutari, M. (2017). Bayesian dirichlet bayesian network scores and the maximum entropy principle. *arXiv preprint arXiv:1708.00689*.

[Shachter, 1986] Shachter, R. (1986). Evaluating influence diagrams. *Operations Research*, 34(6):871–882.

[Shafer, 1996] Shafer, G. (1996). *Probabilistic expert systems*. Society for Industrial and Applied Mathematics.

[Shenoy, 2011] Shenoy, P. (2011). A re-definition of mixtures of polynomials for inference in hybrid Bayesian networks. In *Proceedings of ECSQARU'11*, volume 6717 of *LNCS*, pages 98–109.

[Shenoy, 2012] Shenoy, P. (2012). Two issues in using mixtures of polynomials for inference in hybrid Bayesian networks. *International Journal of Approximate Reasoning*, 53(5):847–866.

[Shenoy and Shafer, 1990] Shenoy, P. and Shafer, G. (1990). Axioms for probability and belief function propagation. In *Proceedings of UAI'90*, pages 169–198.

[Shenoy and West, 2011] Shenoy, P. and West, J. (2011). Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52(5):641–657.

[Shimony, 1994] Shimony, S. E. (1994). Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399 – 410.

[Shioya and Miura, 2011] Shioya, I. and Miura, T. (2011). Square root update acceleration of the em algorithm in gaussian mixture processes. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 167–172.

[Shore and Johnson, 1980] Shore, J. and Johnson, R. (1980). Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on information theory*, 26(1):26–37.

[Shtar'kov, 1987] Shtar'kov, Y. M. (1987). Universal sequential coding of single messages. *Problemy Peredachi Informatsii*, 23(3):3–17.

[Silander et al., 2007] Silander, T., Kontkanen, P., and Myllymäki, P. (2007). On sensitivity of the map bayesian network structure to the equivalent sample size parameter.

[Silander et al., 2018] Silander, T., Leppä-aho, J., Jääsaari, E., and Roos, T. (2018). Quotient normalized maximum likelihood criterion for learning bayesian network structures. In *International Conference on Artificial Intelligence and Statistics*, pages 948–957.

[Silander and Myllymaki, 2012] Silander, T. and Myllymaki, P. (2012). A simple approach for finding the globally optimal bayesian network structure. *arXiv preprint arXiv:1206.6875*.

[Silander et al., 2008] Silander, T., Roos, T., Kontkanen, P., and Myllymäki, P. (2008). Factorized normalized maximum likelihood criterion for learning bayesian network structures. In *Proceedings of the 4th European Workshop on Probabilistic Graphical Models*, pages 257–272.

[Singh and Moore, 2005] Singh, A. P. and Moore, A. W. (2005). *Finding optimal Bayesian networks by dynamic programming*. Citeseer.

[Skaanning et al., 2000] Skaanning, C., Jensen, F. V., and Kjærulff, U. (2000). Printer troubleshooting using bayesian networks. In *Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Intelligent Problem Solving: Methodologies and Approaches*, IEA/AIE '00, pages 367–379, Berlin, Heidelberg. Springer-Verlag.

[Spirtes et al., 2001] Spirtes, P., Glymour, C., and Scheines, R. (2001). *Causation, Prediction and Search*. Bradford Book, 2nd edition.

[Starnes et al., 2010] Starnes, D., Yates, D., and Moore, D. (2010). *The Practice of Statistics*. W. H. Freeman.

[Steck and Tresp, 1999] Steck, H. and Tresp, V. (1999). Bayesian belief networks for data mining. In *Proceedings of the 2. Workshop on Data Mining und Data Warehousing als Grundlage moderner entscheidungsunterstützender Systeme*, pages 145–154. Citeseer.

[Suzuki, 2017] Suzuki, J. (2017). A theoretical analysis of the bdeu scores in bayesian network structure learning. *Behaviormetrika*, 44(1):97–116.

[Szpankowski and Weinberger, 2012] Szpankowski, W. and Weinberger, M. J. (2012). Minimax pointwise redundancy for memoryless models over large alphabets. *IEEE Transactions on Information Theory*, 58(7):4094–4104.

[Teyssier and Koller, 2012] Teyssier, M. and Koller, D. (2012). Ordering-based search: A simple and effective algorithm for learning bayesian networks. *arXiv preprint arXiv:1207.1429*.

[Tsamardinos et al., 2003] Tsamardinos, I., Aliferis, C. F., and Statnikov, A. (2003). Time and sample efficient discovery of markov blankets and direct causal relations. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 673–678. ACM.

[Tsamardinos et al., 2006] Tsamardinos, I., Brown, L. E., and Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78.

[Tsang, 2010] Tsang, R. (2010). Cyberthreats, vulnerabilities and attacks on scada networks.

[van den Eijkhof and Bodlaender, 2002] van den Eijkhof, F. and Bodlaender, H. L. (2002). Safe reduction rules for weighted treewidth. In *Proceedings of WG'02*, volume 2573 of *LNCS*, pages 176–185.

[van Dijk et al., 2003] van Dijk, S., Van Der Gaag, L. C., and Thierens, D. (2003). A skeleton-based approach to learning bayesian networks from data. In *Knowledge Discovery in Databases: PKDD 2003*, pages 132–143. Springer.

[Vandel et al., 2012] Vandel, J., Mangin, B., and De Givry, S. (2012). New local move operators for bayesian network structure learning. *Proceedings of PGM-12, Granada, Spain*.

[Varadhan and Roland, 2008] Varadhan, R. and Roland, C. (2008). Simple and globally convergent methods for accelerating the convergence of any em algorithm. *Scandinavian Journal of Statistics*, 35(2):335–353.

[Verma and Pearl, 1991] Verma, T. and Pearl, J. (1991). Equivalence and synthesis of causal models. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 255–270, New York, NY, USA. Elsevier Science Inc.

[Wermuth, 1980] Wermuth, N. (1980). Linear recursive equations, covariance selection, and path analysis. *Journal of the American Statistical Association*, 75(372):963–972.

[Yaramakala and Margaritis, 2005] Yaramakala, S. and Margaritis, D. (2005). Speculative markov blanket discovery for optimal feature selection. In *Data mining, fifth IEEE international conference on*, pages 4–pp. IEEE.

[Yedidia et al., 2001] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2001). Generalized belief propagation. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 689–695. MIT Press.

[Yuan et al., 2011] Yuan, C., Lim, H., and Littman, M. (2011). Most Relevant Explanation: computational complexity and approximation methods. *Annals of Mathematics and Artificial Intelligence*, 61(3):159–183.

[Yuan and Lu, 2008] Yuan, C. and Lu, T.-C. (2008). A general framework for generating multivariate explanations in bayesian networks. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 1119–1124. AAAI Press.

[Zadeh, 1965] Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8(3):338 – 353.

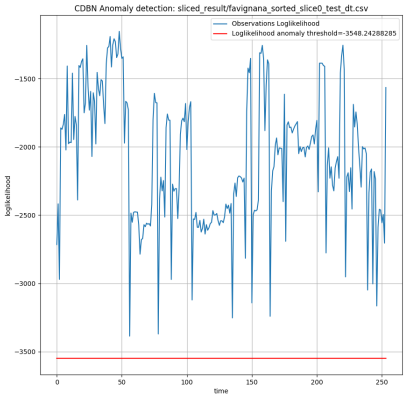
Appendices

Appendix A

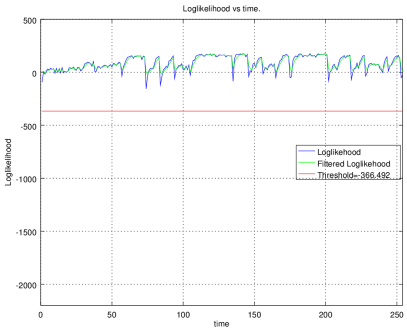
CdBN vs. Kalman Experiments: Detailed Results

In this appendix, we display the plots of the log-likelihoods for every test dataset, from the experiments performed in Section 5.3. This provides more insight on the behaviors of the cdBN and the Kalman Filter anomaly detections. In these plots, the horizontal red lines represent the log-likelihood threshold. So, when the blue curves are below red lines, the algorithms consider such situations as anomalies. Each plot ranges over a whole epoch.

Detailed results on unperturbed dataset $\mathring{\mathcal{D}}_h^T$

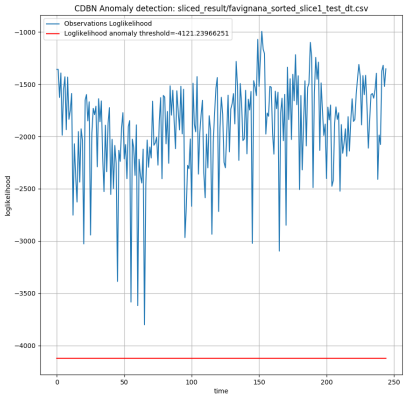


(a) cdBN anomaly detection

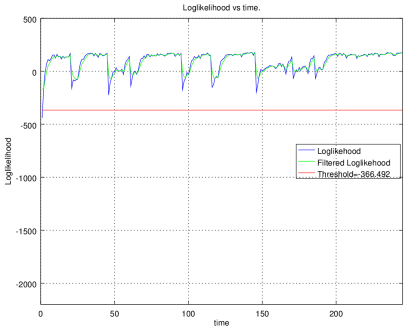


(b) Kalman filter anomaly detection

Figure A.1: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the first epoch.



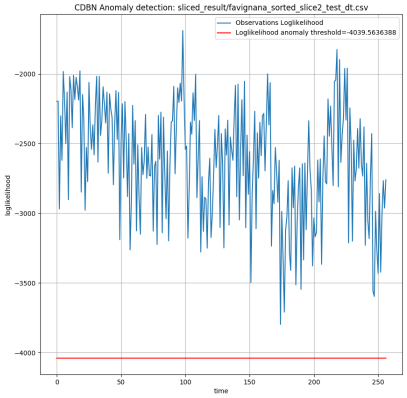
(a) cdBN anomaly detection



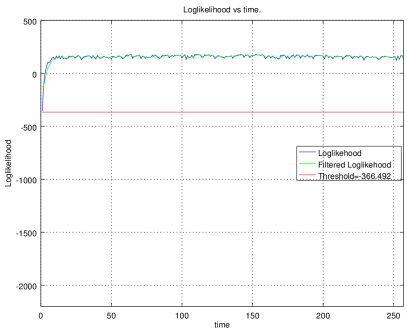
(b) Kalman filter anomaly detection

Figure A.2: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the second epoch.

Here, note that the curves of the likelihoods are much smoother in the Kalman filter. This results from the lowpass filter we used (see Subsection 5.3.3).

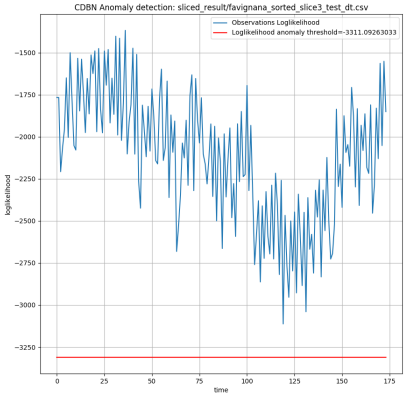


(a) cdBN anomaly detection

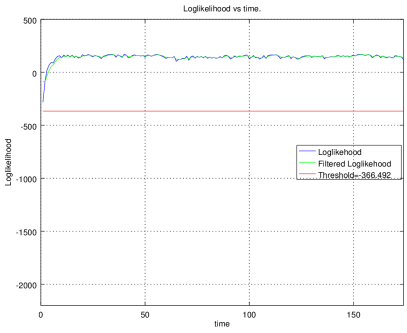


(b) Kalman filter anomaly detection

Figure A.3: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the third epoch.

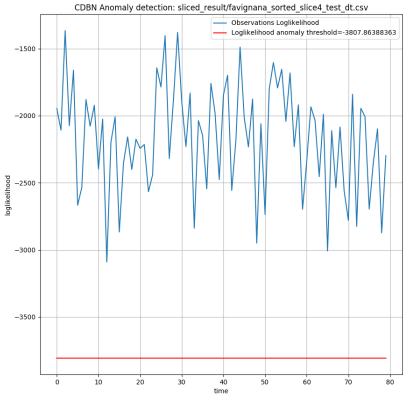


(a) cdBN anomaly detection

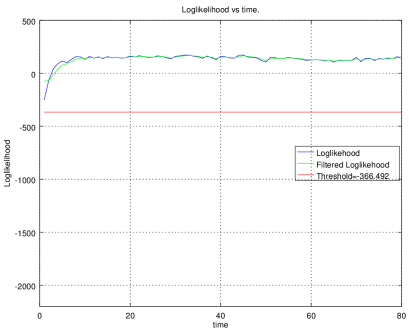


(b) Kalman filter anomaly detection

Figure A.4: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the fourth epoch.

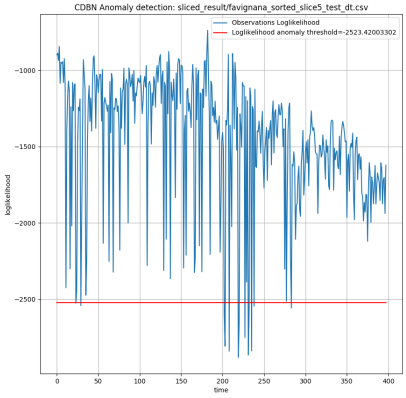


(a) cdBN anomaly detection

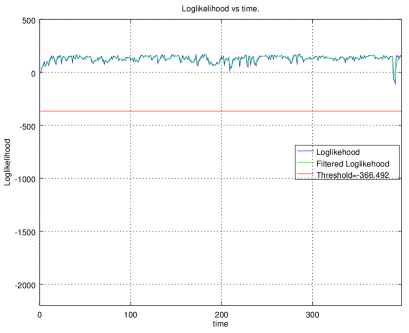


(b) Kalman filter anomaly detection

Figure A.5: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the fifth epoch.



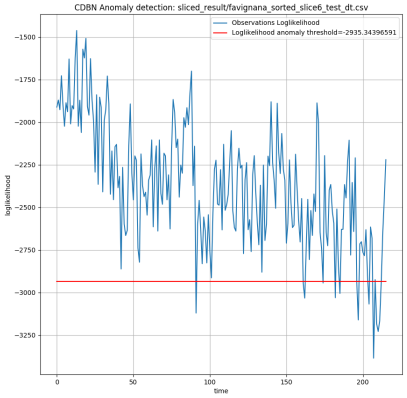
(a) cdBN anomaly detection



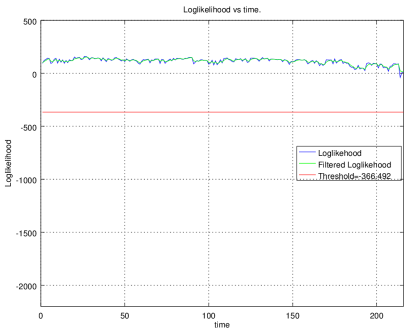
(b) Kalman filter anomaly detection

Figure A.6: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the sixth epoch.

In the sixth epoch, the cdBN is misled several times by likelihoods below the threshold. Therefore, it raises some false alarms. Note however that these likelihoods are not very far from the "red" threshold, which means that the alarm raised by the cdBN is somewhat "weak".

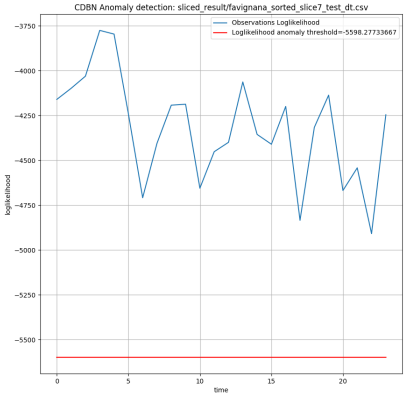


(a) cdBN anomaly detection

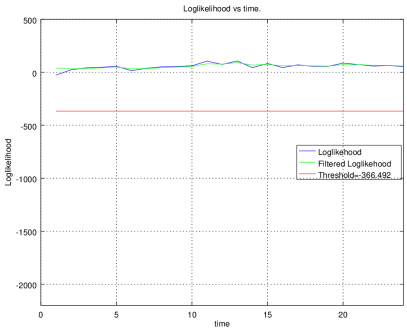


(b) Kalman filter anomaly detection

Figure A.7: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the seventh epoch.



(a) cdBN anomaly detection

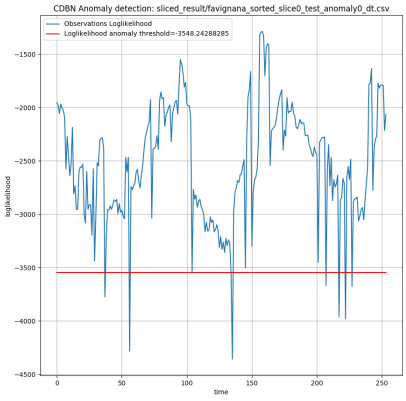


(b) Kalman filter anomaly detection

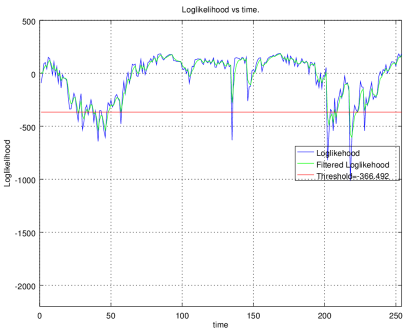
Figure A.8: Anomaly detection over test dataset without perturbations $\mathring{\mathcal{D}}_h^T$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(0)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the whole dataset $\mathring{\mathcal{D}}_h^{T}$.

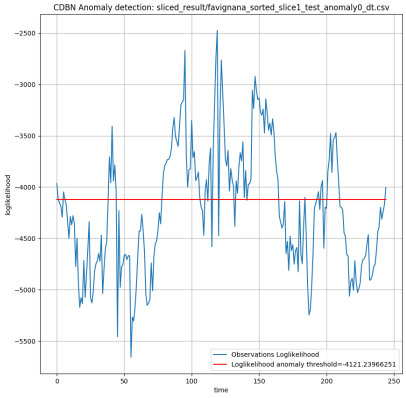


(a) cdBN anomaly detection

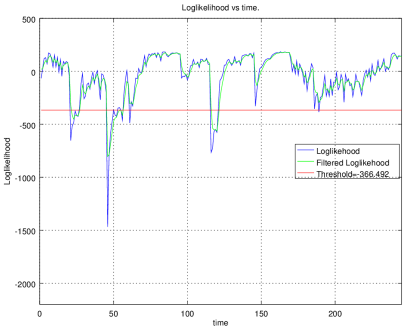


(b) Kalman filter anomaly detection

Figure A.9: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the first epoch.



(a) cdBN anomaly detection

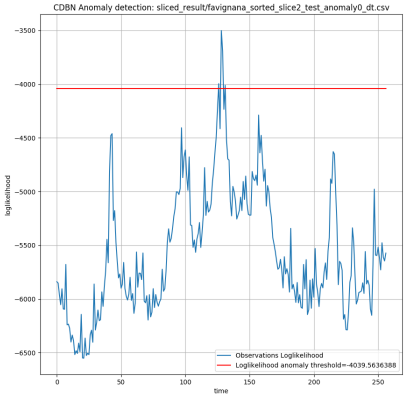


(b) Kalman filter anomaly detection

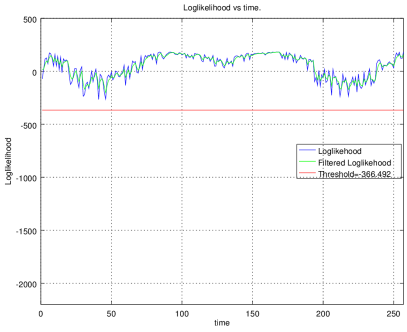
Figure A.10: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the second epoch.

In these figures, both models detect anomalies all over the epochs. However, the cdBN tends to produce likelihoods more often below the threshold than the

Kalman filter. This results in the cdBN being more prone than Kalman to raise alarms.

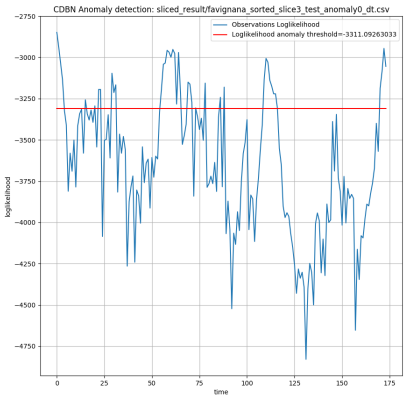


(a) cdBN anomaly detection

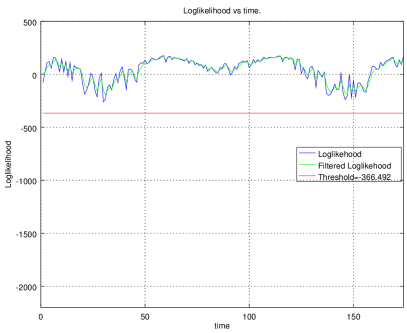


(b) Kalman filter anomaly detection

Figure A.11: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the third epoch.



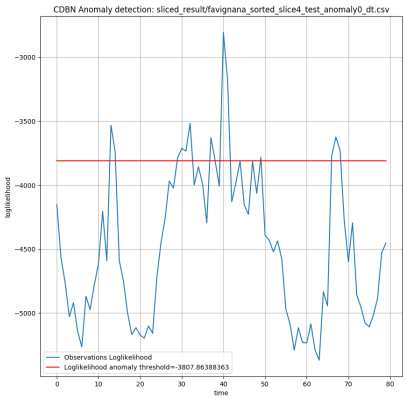
(a) cdBN anomaly detection



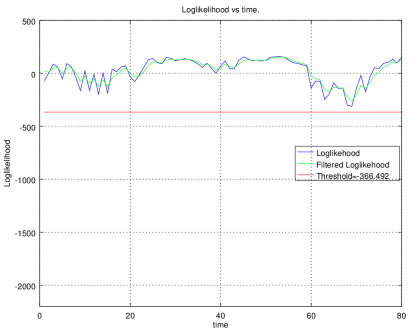
(b) Kalman filter anomaly detection

Figure A.12: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the fourth epoch.

In the third and fourth epochs, the Kalman filter is unable to detect any anomaly whereas the cdBN frequently detects them.

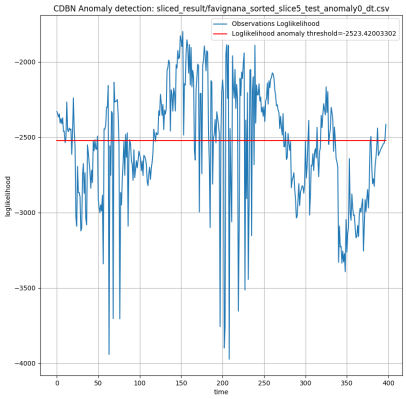


(a) cdBN anomaly detection

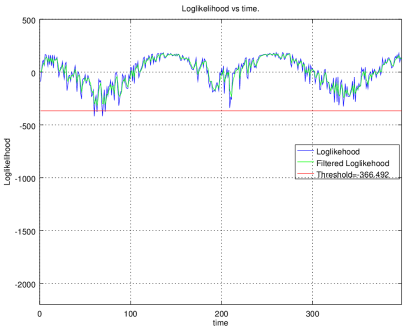


(b) Kalman filter anomaly detection

Figure A.13: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the fifth epoch.

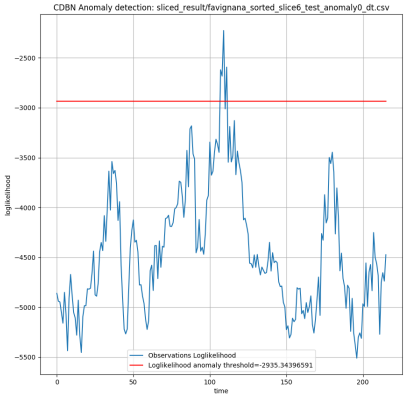


(a) cdBN anomaly detection

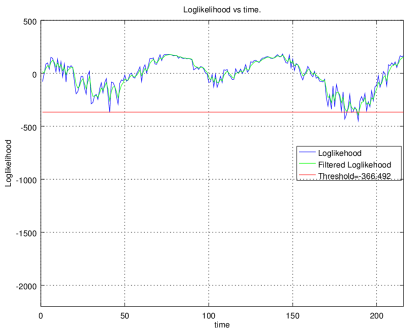


(b) Kalman filter anomaly detection

Figure A.14: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the sixth epoch.

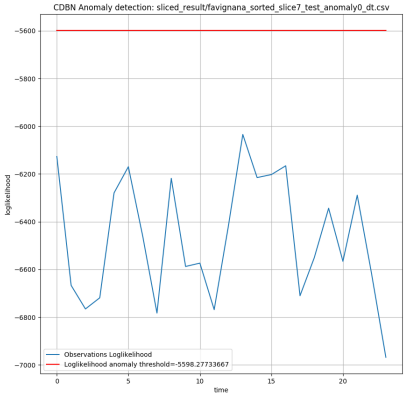


(a) cdBN anomaly detection

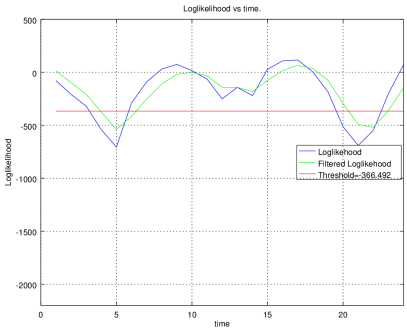


(b) Kalman filter anomaly detection

Figure A.15: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the seventh epoch.



(a) cdBN anomaly detection

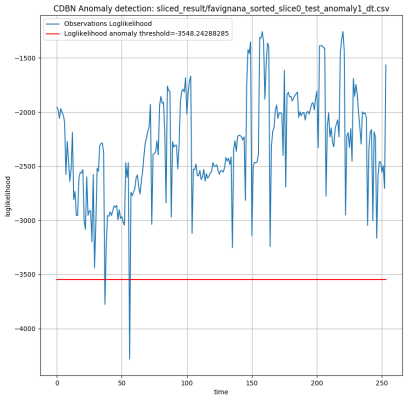


(b) Kalman filter anomaly detection

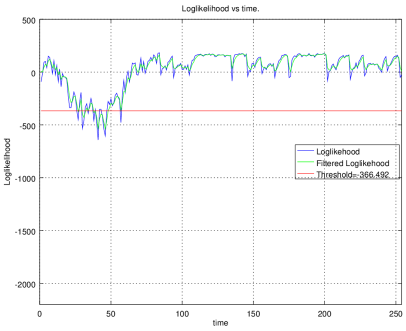
Figure A.16: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(0)}$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(1)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the first third of Dataset $\mathring{\mathcal{D}}_h^{T}$.

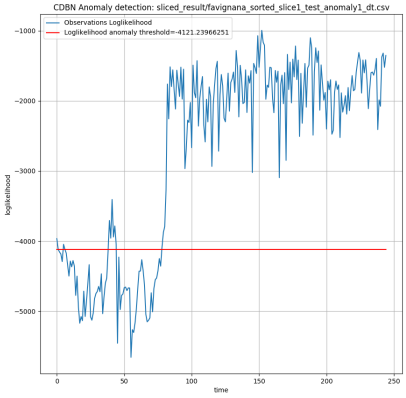


(a) cdBN anomaly detection

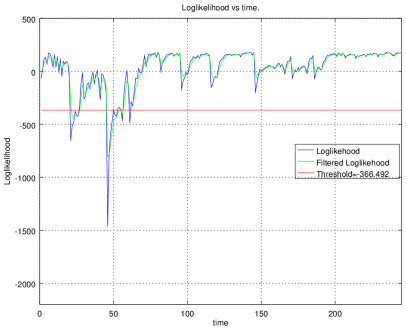


(b) Kalman filter anomaly detection

Figure A.17: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the first epoch.

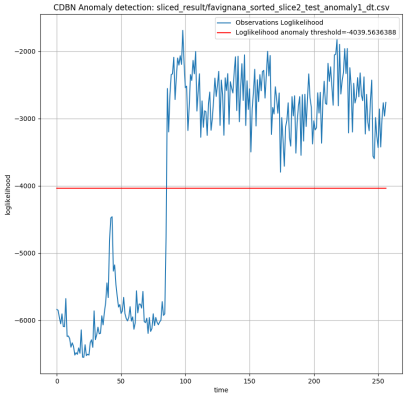


(a) cdBN anomaly detection

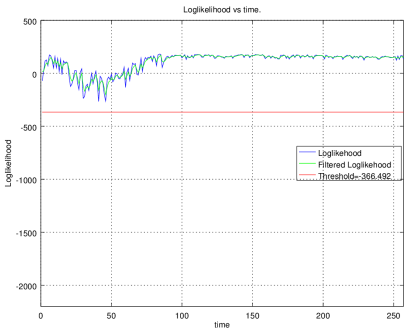


(b) Kalman filter anomaly detection

Figure A.18: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the second epoch.

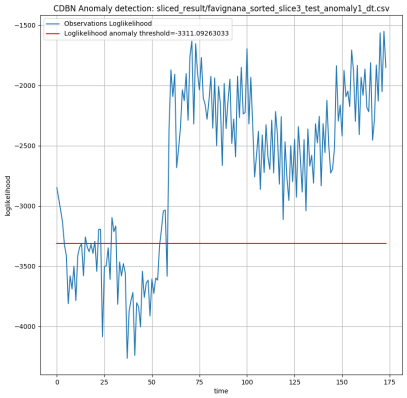


(a) cdBN anomaly detection

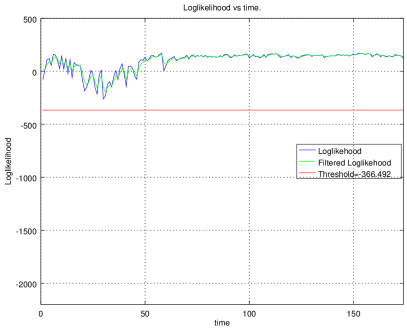


(b) Kalman filter anomaly detection

Figure A.19: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the third epoch.



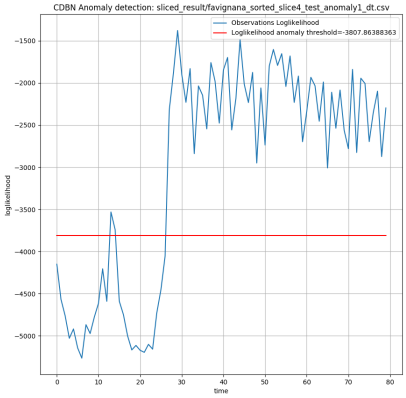
(a) cdBN anomaly detection



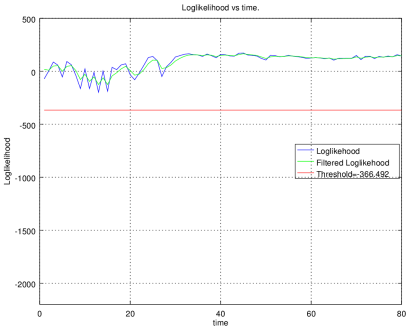
(b) Kalman filter anomaly detection

Figure A.20: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the fourth epoch.

Here, note the ability of the non-stationary cdBN to detect anomalies in the left one third part of the figures (in which there are truly anomalies) and to not detect any anomaly in the two third parts on the right (which actually contain no anomaly).

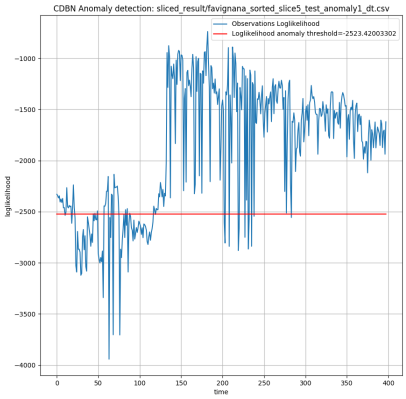


(a) cdBN anomaly detection

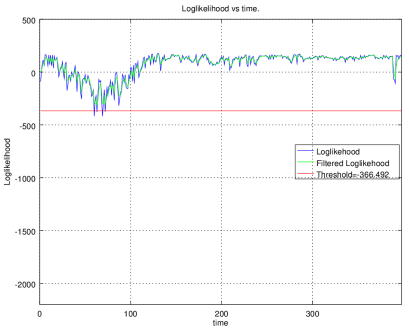


(b) Kalman filter anomaly detection

Figure A.21: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the fifth epoch.

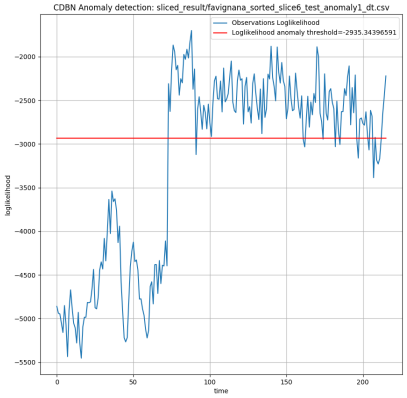


(a) cdBN anomaly detection

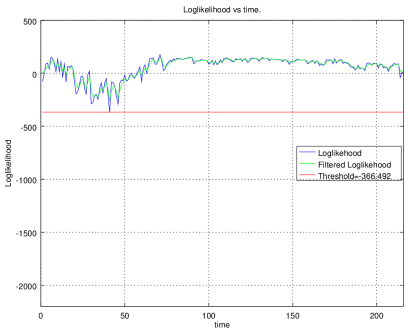


(b) Kalman filter anomaly detection

Figure A.22: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the sixth epoch.

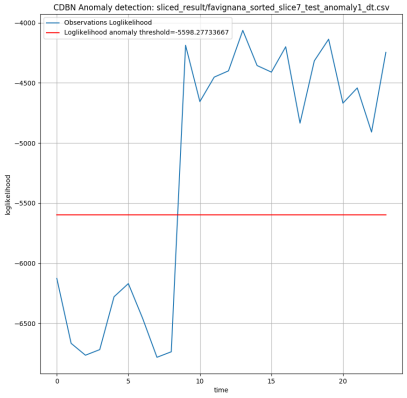


(a) cdBN anomaly detection

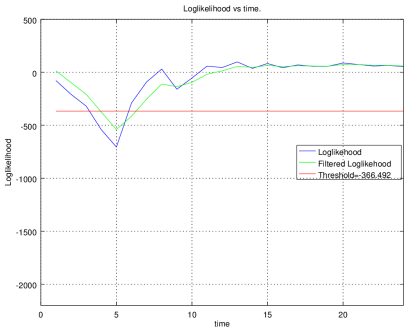


(b) Kalman filter anomaly detection

Figure A.23: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the seventh epoch.



(a) cdBN anomaly detection

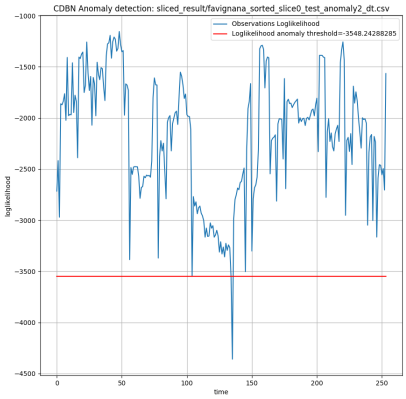


(b) Kalman filter anomaly detection

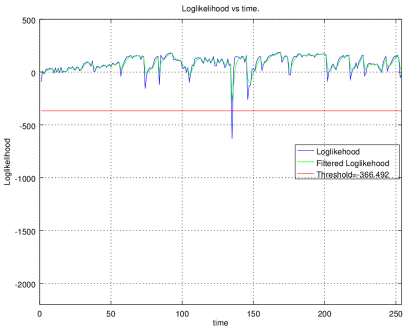
Figure A.24: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(1)}$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(2)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the second third of dataset $\mathring{\mathcal{D}}_h^{T}$.

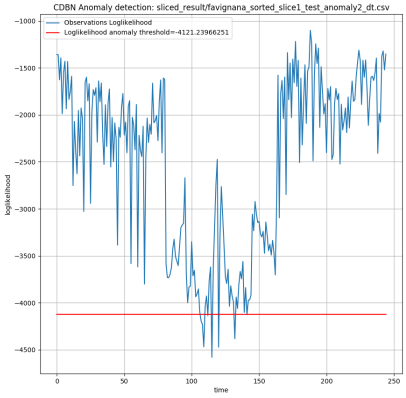


(a) cdBN anomaly detection

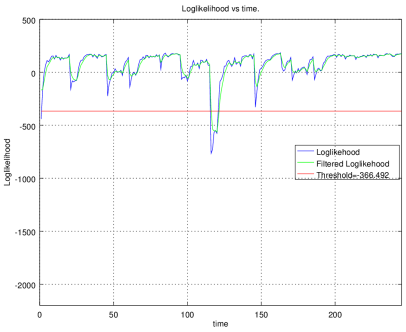


(b) Kalman filter anomaly detection

Figure A.25: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the first epoch.

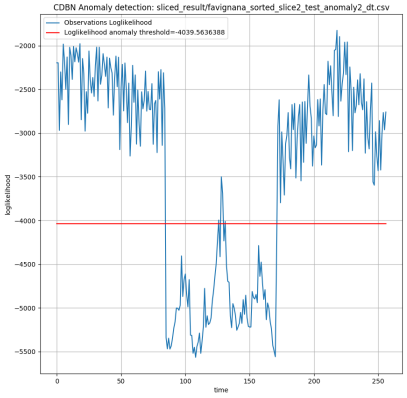


(a) cdBN anomaly detection

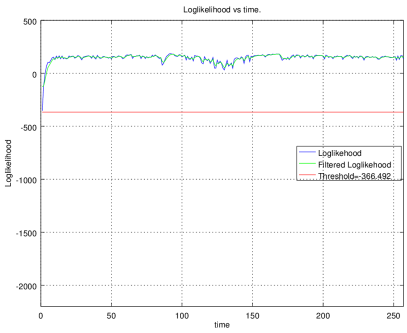


(b) Kalman filter anomaly detection

Figure A.26: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the second epoch.

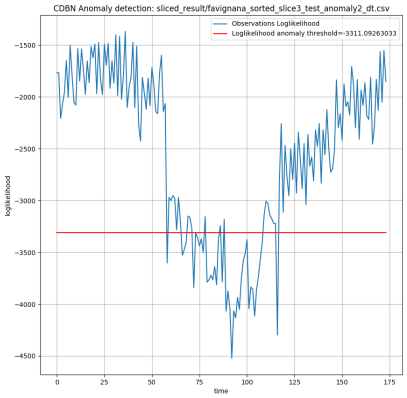


(a) cdBN anomaly detection

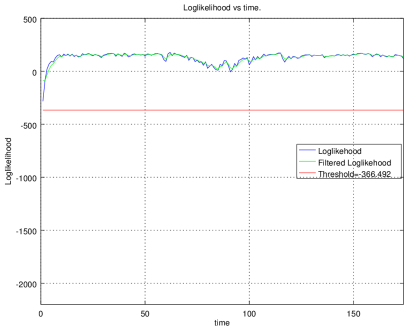


(b) Kalman filter anomaly detection

Figure A.27: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the third epoch.

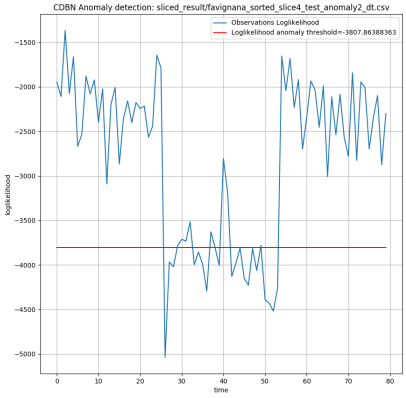


(a) cdBN anomaly detection

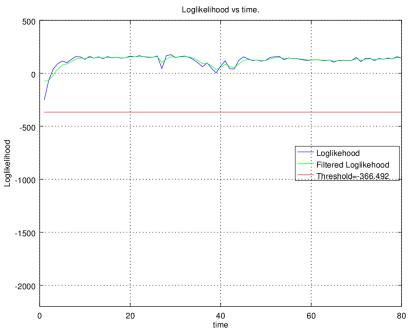


(b) Kalman filter anomaly detection

Figure A.28: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the fourth epoch.

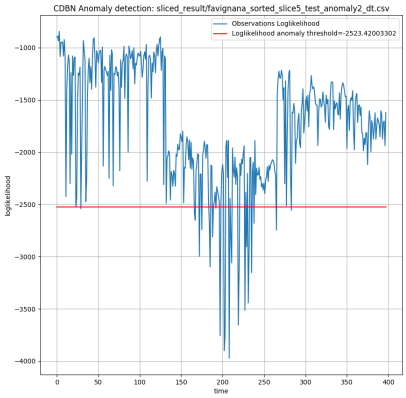


(a) cdBN anomaly detection

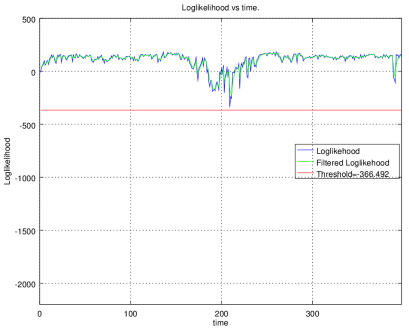


(b) Kalman filter anomaly detection

Figure A.29: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the fifth epoch.

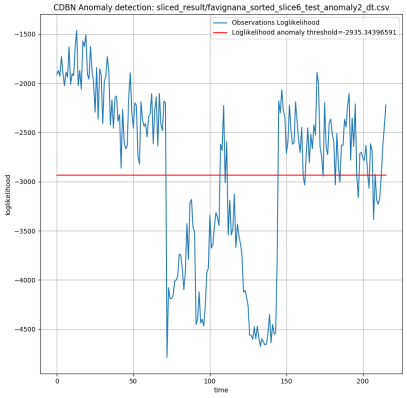


(a) cdBN anomaly detection

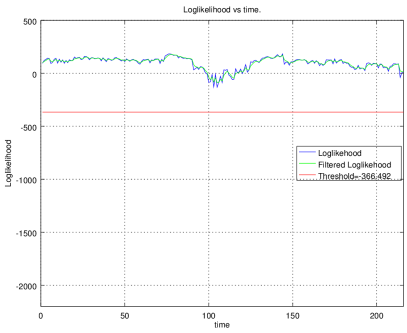


(b) Kalman filter anomaly detection

Figure A.30: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the sixth epoch.

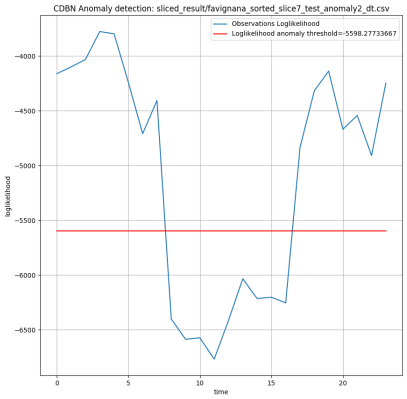


(a) cdBN anomaly detection

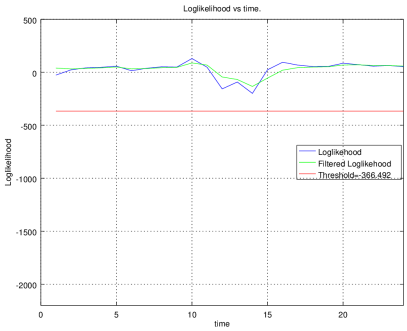


(b) Kalman filter anomaly detection

Figure A.31: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the seventh epoch.



(a) cdBN anomaly detection

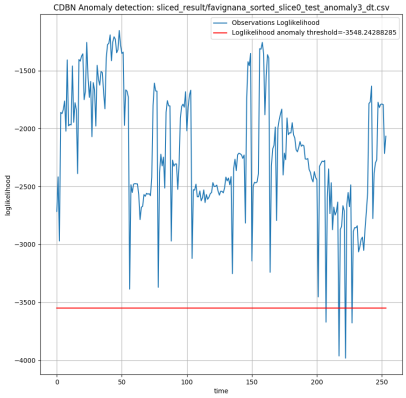


(b) Kalman filter anomaly detection

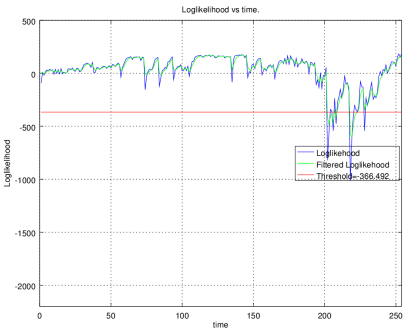
Figure A.32: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(2)}$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(3)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the last third of dataset $\mathring{\mathcal{D}}_h^{T}$.

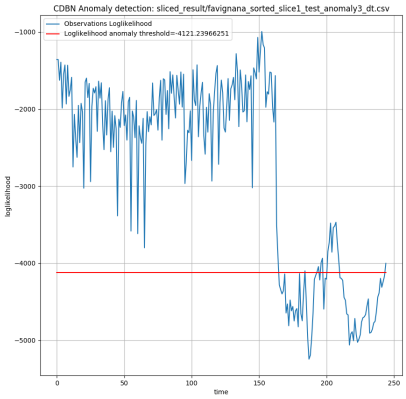


(a) cdBN anomaly detection

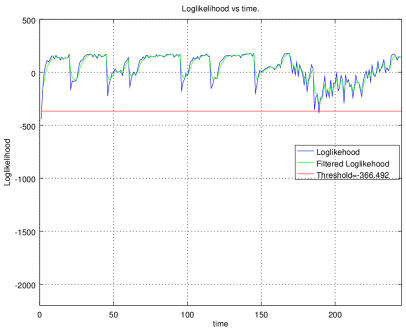


(b) Kalman filter anomaly detection

Figure A.33: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the first epoch.

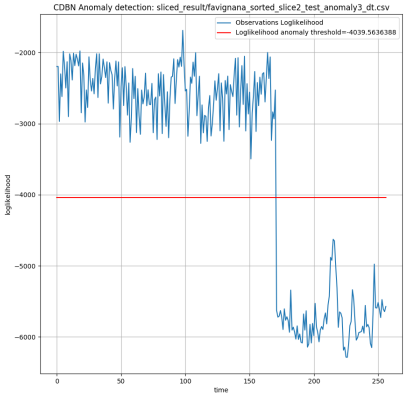


(a) cdBN anomaly detection

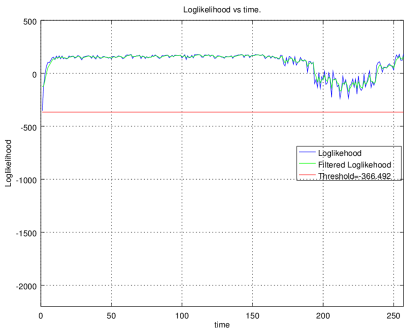


(b) Kalman filter anomaly detection

Figure A.34: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the second epoch.

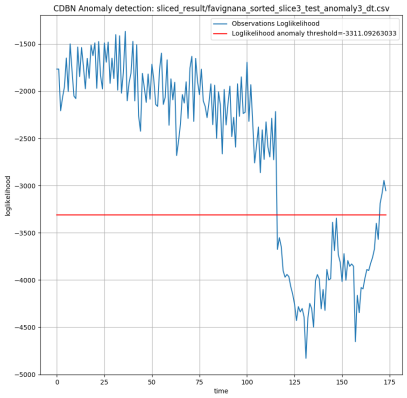


(a) cdBN anomaly detection

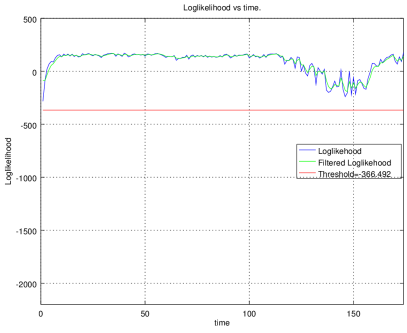


(b) Kalman filter anomaly detection

Figure A.35: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the third epoch.

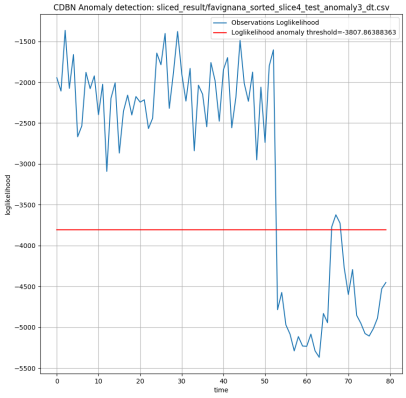


(a) cdBN anomaly detection

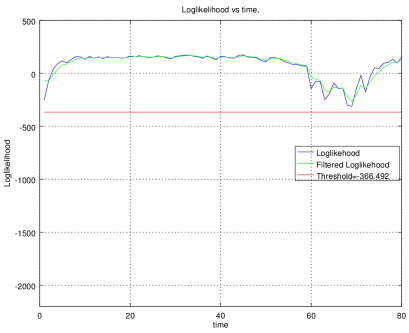


(b) Kalman filter anomaly detection

Figure A.36: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the fourth epoch.

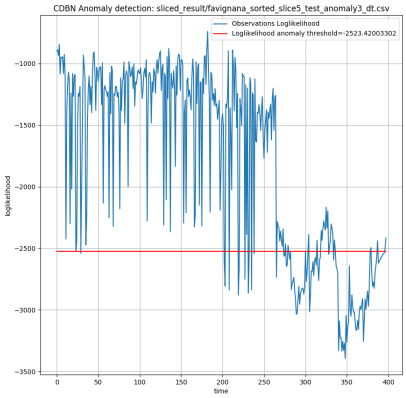


(a) cdBN anomaly detection

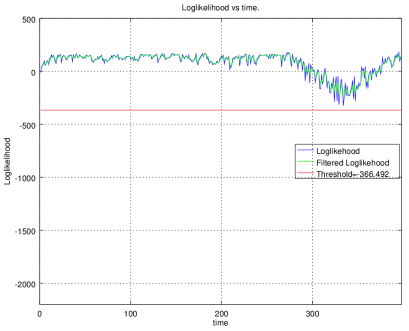


(b) Kalman filter anomaly detection

Figure A.37: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the fifth epoch.

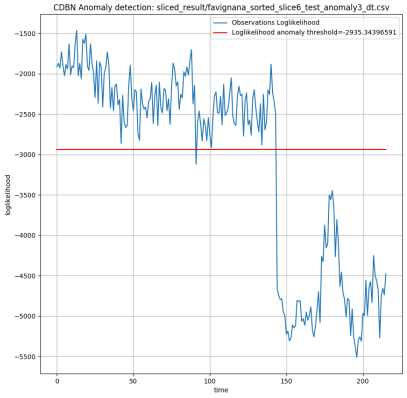


(a) cdBN anomaly detection

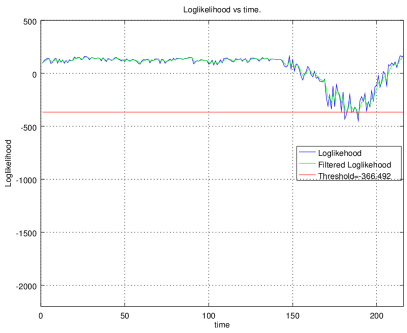


(b) Kalman filter anomaly detection

Figure A.38: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the sixth epoch.

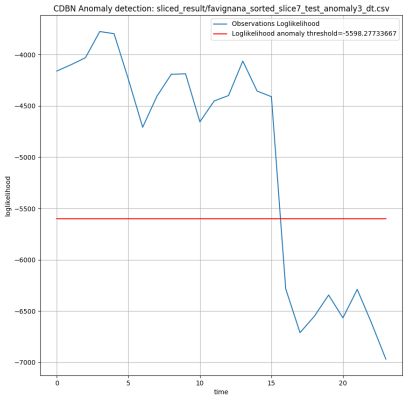


(a) cdBN anomaly detection

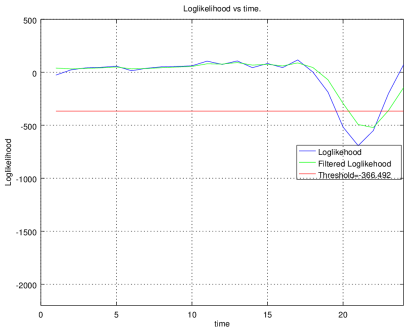


(b) Kalman filter anomaly detection

Figure A.39: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the seventh epoch.



(a) cdBN anomaly detection

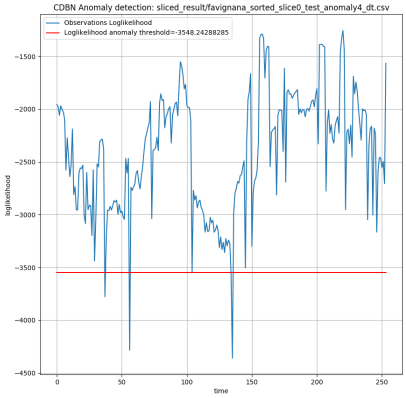


(b) Kalman filter anomaly detection

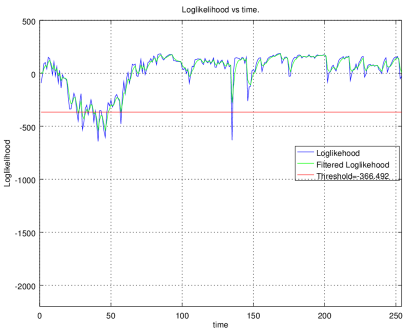
Figure A.40: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(3)}$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(4)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the first two thirds of dataset $\mathring{\mathcal{D}}_h^{T}$.

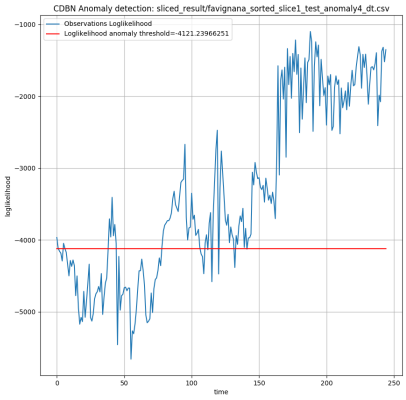


(a) cdBN anomaly detection

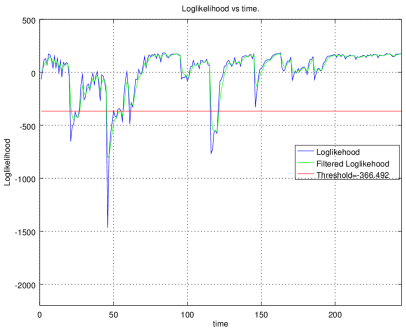


(b) Kalman filter anomaly detection

Figure A.41: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the first epoch.

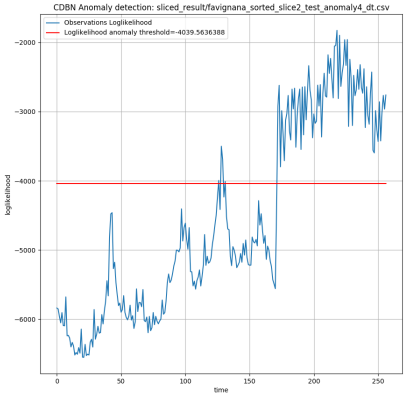


(a) cdBN anomaly detection

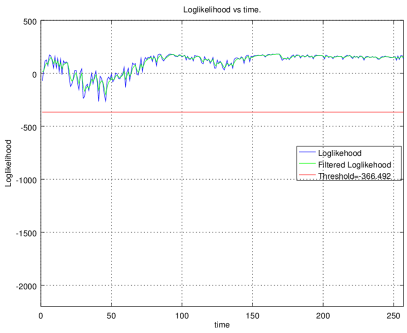


(b) Kalman filter anomaly detection

Figure A.42: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the second epoch.

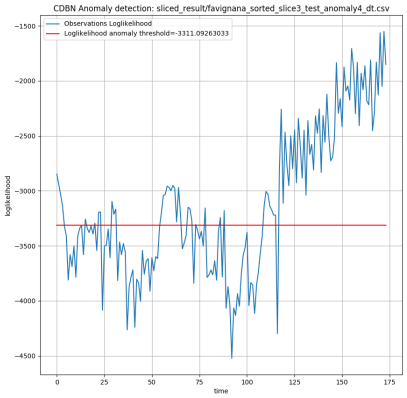


(a) cdBN anomaly detection

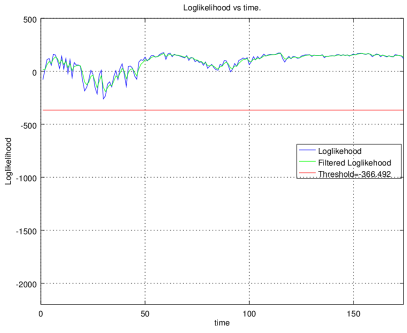


(b) Kalman filter anomaly detection

Figure A.43: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the third epoch.

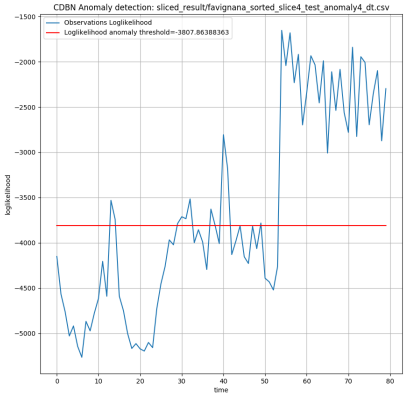


(a) cdBN anomaly detection

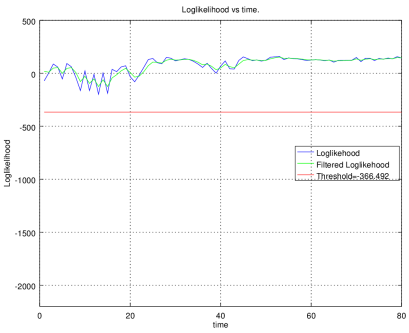


(b) Kalman filter anomaly detection

Figure A.44: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the fourth epoch.

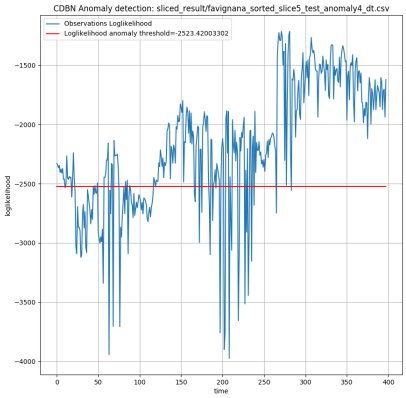


(a) cdBN anomaly detection

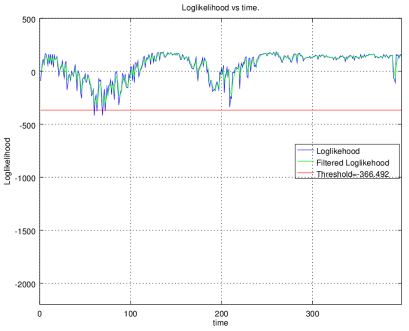


(b) Kalman filter anomaly detection

Figure A.45: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the fifth epoch.

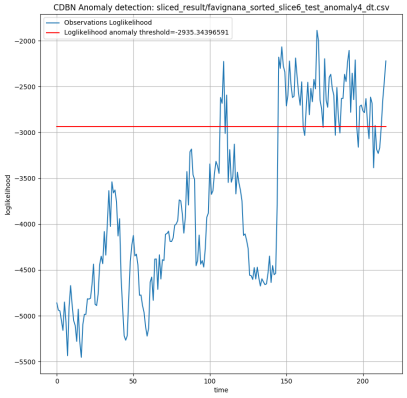


(a) cdBN anomaly detection

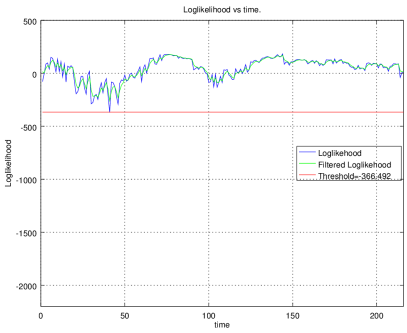


(b) Kalman filter anomaly detection

Figure A.46: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the sixth epoch.

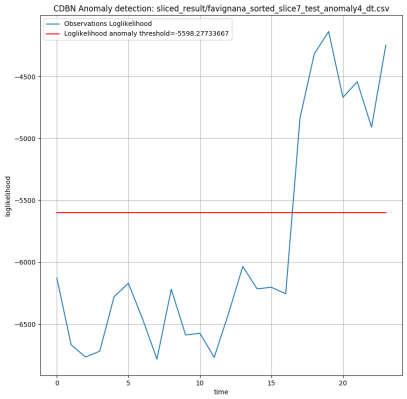


(a) cdBN anomaly detection

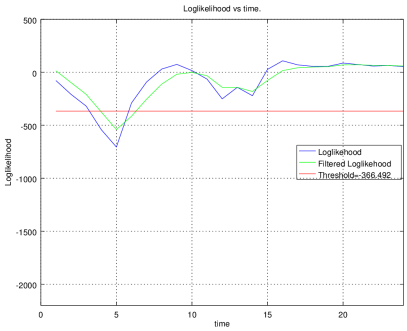


(b) Kalman filter anomaly detection

Figure A.47: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the seventh epoch.



(a) cdBN anomaly detection

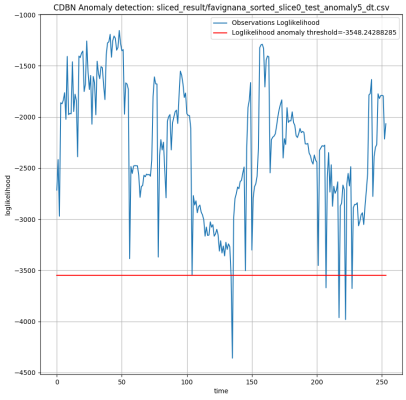


(b) Kalman filter anomaly detection

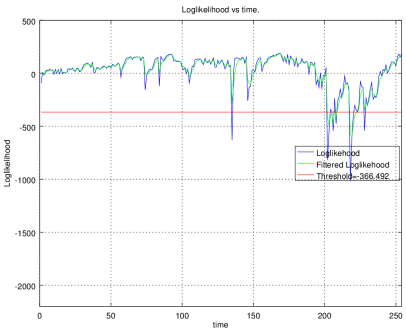
Figure A.48: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(4)}$ in the eighth epoch.

Detailed results on perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$

In Dataset $\mathring{\mathcal{D}}_h^{T(5)}$, Interval $[t_0, t_f]$ over which perturbations are inserted corresponds to the last two thirds of dataset $\mathring{\mathcal{D}}_h^{T}$.

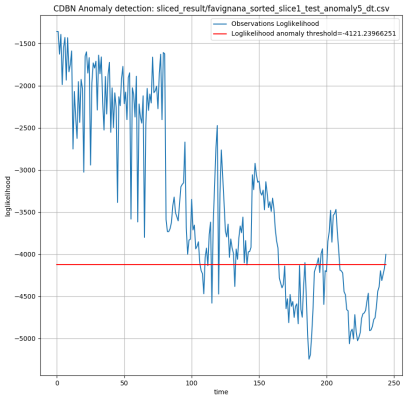


(a) cdBN anomaly detection

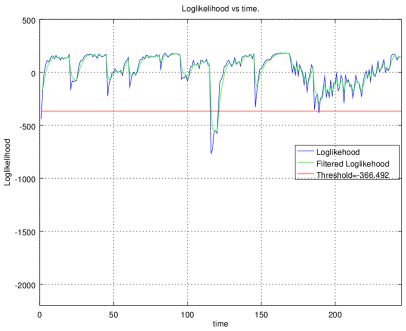


(b) Kalman filter anomaly detection

Figure A.49: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the first epoch.

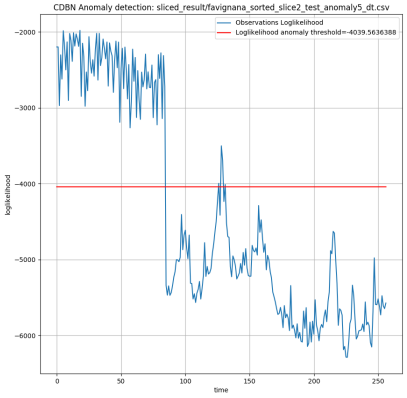


(a) cdBN anomaly detection

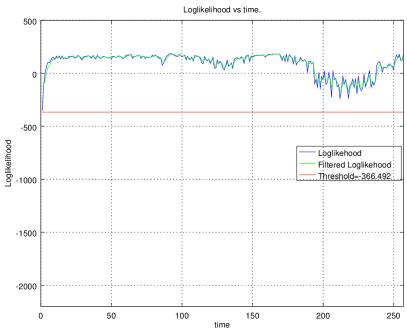


(b) Kalman filter anomaly detection

Figure A.50: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the second epoch.

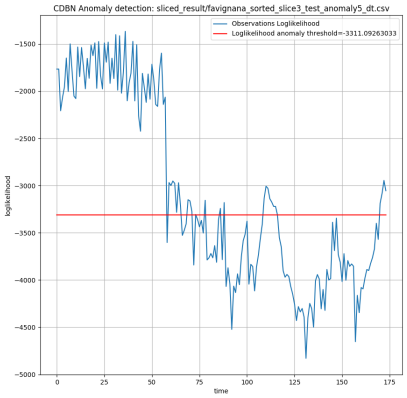


(a) cdBN anomaly detection

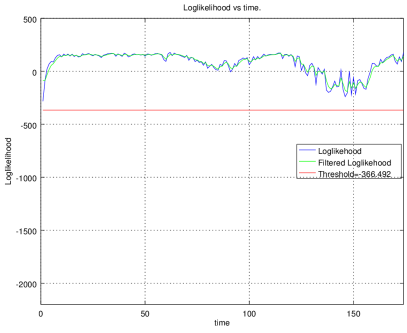


(b) Kalman filter anomaly detection

Figure A.51: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the third epoch.

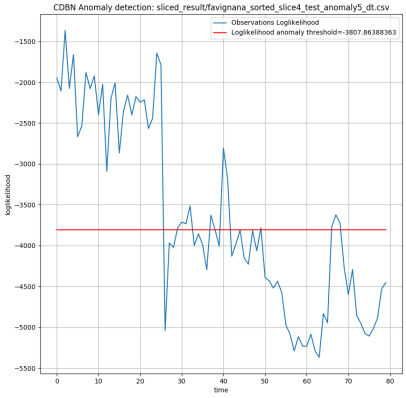


(a) cdBN anomaly detection

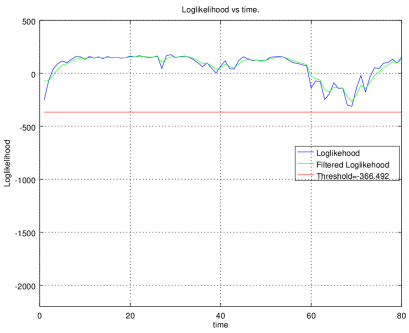


(b) Kalman filter anomaly detection

Figure A.52: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the fourth epoch.

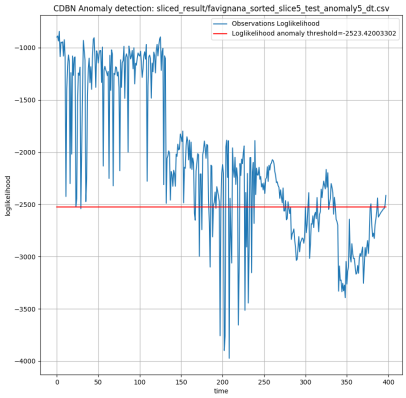


(a) cdBN anomaly detection

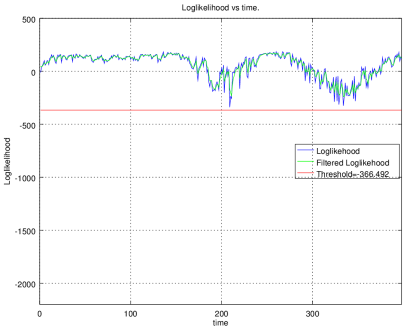


(b) Kalman filter anomaly detection

Figure A.53: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the fifth epoch.

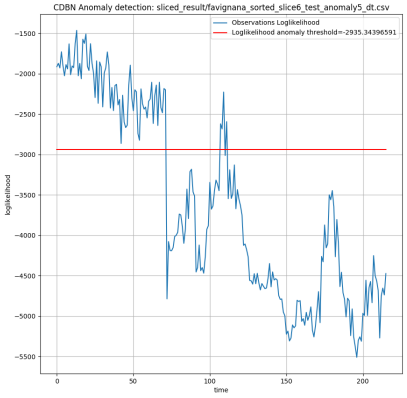


(a) cdBN anomaly detection

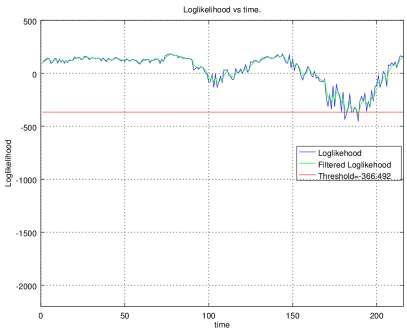


(b) Kalman filter anomaly detection

Figure A.54: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the sixth epoch.

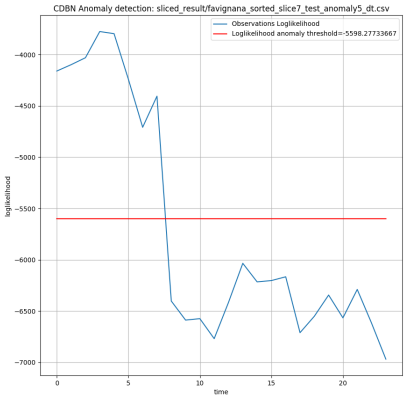


(a) cdBN anomaly detection

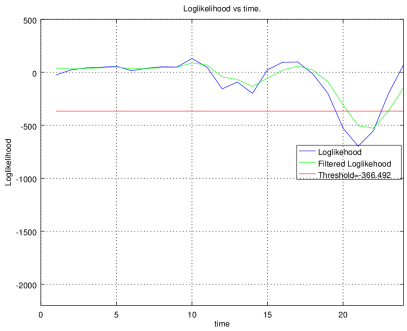


(b) Kalman filter anomaly detection

Figure A.55: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the seventh epoch.



(a) cdBN anomaly detection



(b) Kalman filter anomaly detection

Figure A.56: Anomaly detection over perturbed dataset $\mathring{\mathcal{D}}_h^{T(5)}$ in the eighth epoch.