Adapting the Prerequisite Structure to the Learner in Student Modeling

Olivier Allègre

Sorbonne Université

Ecole doctorale Informatique, Télécommunications et Electronique

Laboratoire d'informatique de Paris 6 / MOCAH

Adapting the Prerequisite Structure to the Learner in Student Modeling

Par Olivier Allègre

Thèse de doctorat d'informatique

Dirigée par Vanda Luengo

Co-encadrée par Amel Yessad

Présentée et soutenue publiquement le 17 mai 2023

Devant un jury composé de :

- Michel Desmarais, Professeur des universités, Rapporteur (Président du jury)
- Olga C. Santos, Professeure des universités, Rapportrice
- Pierre-Henri Wuillemin, Maître de conférences, Examinateur
- Jill-Jênn Vie, Chargé de Recherche, Examinateur
- Vanda Luengo, Professeure des universités, Directrice de thèse
- Amel Yessad, Chargée de Recherche, Co-encadrante

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Olivier Allègre
March 2023

Acknowledgments

I would like to take this opportunity to express my sincere appreciation to all the individuals who have supported me throughout my Ph.D. Without their invaluable support, guidance, and encouragement, this research would not have been possible.

First and foremost, I want to express my gratitude to my supervisors, Vanda Luengo and Amel Yessad, for their guidance throughout the entire duration of my Ph.D. Their expertise and contribution have been instrumental in achieving the goals of this research, and their unwavering support allowed me to continue my thesis despite the contractual hurdles I faced. I also want to thank the jury members, Olga C. Santos, Michel Desmarais, Pierre-Henri Wuillemin, and Jill-Jênn Vie, for their time and dedication in reading this manuscript and their insightful feedback. In particular, I want to thank Pierre-Henri for his support in my work with Bayesian networks, which helped me to improve the quality of my research. I would like to acknowledge my colleagues at MOCAH, in particular my fellow Ph.D. students, Mélina, Jérôme, and Thomas, for their faithful and caring companionship during this challenging journey, venting our frustrations together. Their encouragement and enthusiasm have made this journey a truly memorable experience. I wish them all the success for the rest of their work.

Then, I would like to thank my ex-colleagues and friends at Kartable, especially Béryl, Marianne, and Mélissa, with whom I have shared plenty of unforgettable adventures, and François and Thibaut, who have quenched my thirst for science. I also want to thank Sarah and Julien. Even if the pandemic situation forced you to make heartbreaking choices, this work would not have been possible without you. I will not forget it. Finally, I want to thank all the high-standard exercise writers who produced an outstanding quantity of exercises for Kartable. Mathieu, Matthieu, Arno, Jérémy, and Corentin, I hope the soup was succulent.

I am deeply grateful to my parents, Isabelle and Vincent, for their love and encouragement throughout my academic journey, to my brother Thibaut, who is next on the list, and to all my family. Their support has been a constant source of motivation. I also want to thank my parents-in-law, Catherine and Cédric, who endured my headaches while they did not ask.

I also want to express my heartfelt gratitude to my dear friends. First, a huge

thanks to those who stayed in Paris these last years and sometimes had to listen to my complaints these last years: Paul, Charlène, Christophe, Corentin, Emmeline, Romain, Arno, Matthieu (yes, I count you in), Jules (yes, you also), Jérémy, Cyrielle, Antoine, Raphaël. Even if you were geographically further away, thank you also to Mathieu, Franz, Sofia, Pierre, Claire, Iris, Tom, Julie, Roxane, and Charlotte. Love you, guys!

Last but not least, I want to give a special thanks to my fiancée Juliette who has been my rock and my biggest supporter throughout this entire journey. Her love and patience have been the foundation of my success. Her trust in me gave me the confidence I needed. I cannot wait for next summer to become fully hers.

Abstract

Data-driven learner models aim to represent and understand students' knowledge and other meta-cognitive characteristics to support their learning by making predictions about their future performance. Learner modeling can be approached using various complex system models, each providing a different perspective on the student and the learning process. Knowledge-enhanced machine learning techniques, such as Bayesian networks, are particularly well suited for incorporating domain knowledge into the learner model, making them a valuable tool in student modeling.

This work explores the modeling and the potential applications of a new framework called Embedding Prerequisite Relationships In Student Modeling (E-PRISM), which includes a learner model based on Dynamic Bayesian Networks (DBNs). It uses a new architecture for Bayesian networks that rely on the clause of Independence of Causal Influences (ICI), which reduces the number of parameters in the network and allows enhanced interpretability. The study examines the strengths of E-PRISM, including its ability to consider the prerequisite structure between knowledge components, its limited number of parameters, and its enhanced interpretability. The study also introduces a novel approach for approximate inference in large ICI-based Bayesian networks, as well as a performant parameter learning algorithm in ICI-based Bayesian networks. Overall, the study demonstrates the potential of E-PRISM as a promising tool for discovering the prerequisite structure of domain knowledge that may be adapted to the learner with the perspective of improving the outer-loop adaptivity.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

$e\Delta$ E-PRISM learner models. 140, 142, 143, 149–151, 154, 156, 158–161, 164, 168, 169, 174

2-TBN Two-Timeslices Bayesian Network. 53

ACC Accuracy. 138

AI Artificial Intelligence. 6

AIED Artificial Intelligence in Education. 3

AUC Area Under the ROC Curve. 20, 138

BGS Blocking Gibbs Sampling. 101, 112, 124, 134

BKT Bayesian Knowledge Tracing. 26, 91, 97, 144, 185

BN Bayesian Network. 14, 182

BNs Bayesian Networks. 10, 25, 32, 37, 39, 48, 55, 57, 84

CbKST Competence-based Knowledge Space Theory. 31, 79

Cifre Convention industrielle de formation par la recherche. 5

CPD Conditional Probability Distribution. 10, 14, 42, 182, 183

CPDs Conditional Probability Distributions. 43, 57, 80, 84, 183

CPTs Conditional Probability Tables. 45, 65

CSI Context-Specific Independence. 45

DAG Direct Acyclic Graph. 48, 85

DBN Dynamic Bayesian Network. 27, 34, 101

DBNs Dynamic Bayesian Networks. vii, 52, 84, 91, 120

DINA Deterministic Input Noisy And. 25

DINO Deterministic Input Noisy Or. 25

DKT Deep Knowledge Tracing. 28

E-PRISM Embedding Prerequisite Relationships In Student Modeling. vii, 9, 77, 98, 100, 182

EDM Educational Data Mining. 3

EdTech Education Technologies. 3, 4

EM Expectation-Maximization. 70

FOL First-Order Logic. 55

HMM Hidden Markov Model. 54

i.i.d. Independent and Identically Distributed. 106

ICI Independence of Causal Influences. vii, 9, 25, 46, 86, 182

ILP Inductive Logic Programming. 67

IRT Item Response Theory. 22

ITS Intelligent Tutoring Systems. 4, 12, 77, 98, 136

KC Knowledge Component. 5, 13, 182

KCs Knowledge Components. 1, 31, 78, 98, 136

KL Kullback-Leibler. 59, 107, 126, 128

KLI Knowledge-Learning-Instruction. 13, 78

KRR Knowledge Representation and Reasoning. 5, 6, 37, 39, 55, 65

KST Knowledge Space Theory. 31, 79

LAK Learning Analytics and Knowledge. 3, 31

LBP Loopy-Belief Propagation. 60

LFA Learning Factor Analysis. 23

MAE Mean Absolute Error. 138

MAP Maximum A Posteriori. 55, 56, 69

MAR Missing At Random. 70

MCAR Missing Completly At Random. 70

MCEM Monte-Carlo Expectation Maximization. 72, 184

MCMC Markov Chain Monte-Carlo. xvii, 62, 105, 123

MH Metropolis-Hastings. 62

MIRT Multidimensional Item Response Theory. 23

MLE Maximum Likelihood Estimate. 68

MNAR Missing Not At Random. 70

MOOC Massively Open Online Course. 4

NIDA Noisy Input Deterministic And. 25

NLL Negative Log-Likelihood. 126, 127

PFA Performance Factor Analysis. 23

PGM Probabilistic Graphical Model. 48

PGMs Probabilistic Graphical Models. 38, 39, 42, 57, 65

PILP Probabilistic Inductive Logic Programming. 67

POKS Partial-Order Knowledge Spaces. 32

PRMs Probabilistic Relational Models. 67

RBNs Relational Bayesian networks. 67

RMSE Root Mean Square Error. 20, 138

ROC Receiver Operating Characteristics. 138

SAKT Self-Attentive Knowledge Tracing. 28

SEM Stochastic Expectation-Maximization. 72

SRL Statistical Relational Learning. 67

TEL Technology-Enhanced Learning. 3, 4, 12

VE Variable Elimination. 58

xAI eXplainable Artificial Intelligence. 8

Disclaimer

In this work, we use a specific notation.

The $\mathfrak{fraktur}$ font for uppercase corresponds to Knowledge Components (KCs). For instance, $\mathfrak{X}$ represents a KC.

The lowercase normal font is dedicated to scalars, and the uppercase normal font is dedicated to random variables. For example, α is a scalar, and X is a random variable.

The boldness of the font indicates whether the object is in a vector. $\mathfrak{Pa}_{\mathfrak{X}}$ is a vector of KCs, $\mathbf{X}$ a vectors of random variables, and $\boldsymbol{\theta}$ a vector of scalars.

Chapter 1

Introduction

Education plays a central role in society as it is the foundation upon which individuals build their futures. It equips people with the necessary knowledge and skills to succeed personally and professionally. Furthermore, education is crucial in fostering social cohesion and promoting social mobility. The motivation for writing a Ph.D. thesis on learner modeling is to contribute to the development of more effective and personalized educational systems. By leveraging learner modeling techniques, it is possible to create more understandable models of individual learners, leading to more effective teaching strategies and providing insights into the learning process.

1.1 On the educational context

First, we provide an overview of the educational context in which this Ph.D. thesis is situated. This includes a discussion of the relevant research domains related to our work and the development of Technology-Enhanced Learning (TEL) systems central to our research. In addition, we present a quick overview of the EdTech (for Education Technologies) ecosystem and explore the particular context of this research, which the pandemic has particularly impacted.

1.1.1 An interdisciplinary research domain

The educational context is a highly interdisciplinary field that brings together various disciplines, such as computer science, cognitive science, education, psychology, and more. This interdisciplinarity allows for a comprehensive understanding of complex learning processes and the development of effective educational technologies. The use of data in the educational context has given rise to new research domains, such as Educational Data Mining (EDM), Learning Analytics and Knowledge, and Artificial Intelligence in Education (AIED). EDM is concerned with developing and applying data mining techniques to automate the discoveries of learners and their contexts, focusing on automation and prediction [LLB18]. At the same time, LAK focuses on the measurement, collection, analysis, and reporting of data about learners and their contexts for purposes of leveraging human judgment on learning and the environments in which it occurs, with a focus on visualization [LLB18]. Additionally, the

AIED community focuses on the design and development of agents and tutors to support and enhance the learning process [LLB18]. Their research themes include agents and tutors, with a focus on using artificial intelligence to improve educational experiences. These research domains seek to address the challenges of education and learning in the digital age, with each community bringing unique perspectives and approaches to enhance the quality and effectiveness of educational experiences for learners.

1.1.2 TEL systems

Significant changes are happening to the way education is delivered, leading to an increased need for and use of Technology-Enhanced Learning (TEL) systems. As a result, the EDM, LAK, and AIED domains have also seen growth and further development. EDM models have become more widely used as the basis for automated adaptation in TEL environments, such as Intelligent Tutoring Systems (ITS), to support remote learning. On the other hand, LAK models, which are designed to inform and empower educators and learners through personalized and effective feedback, have become increasingly relevant in this context.

For instance, Cognitive Tutor, an ITS developed by Carnegie Mellon University in the 1990s [ACKP95], was designed to teach math and science concepts to high school students and has been used extensively in classrooms around the world. One of the key features of Cognitive Tutor is its use of cognitive models to represent the learner's knowledge and to adapt the instruction to the learner's needs. In addition to its use in classrooms, Cognitive Tutor has also been used in research studies to investigate the effectiveness of different teaching approaches and to develop new insights into how students learn [AMRK06, RAKC07]. Cognitive Tutor is a notable example of an ITS agent and illustrates the transformation of teaching practices.

1.1.3 EdTech ecosystem

While universities and laboratories have produced the majority of ITS for research purposes, numerous firms have developed new solutions proposed to schools, teachers, or students. The EdTech (for Education Technologies) market size was valued at US$ 183.4 billion in 2021 [edt].

On the one hand, some non-profit companies provide ITS for free and live through donations. For instance, Khan Academy[1] is a free online learning platform that offers a variety of educational resources and tools. It was founded in 2008 by Salman Khan with the goal of providing accessible, high-quality educational content and resources to students around the world. Coursera[2] is an online learning platform that offers MOOCs from universities around

[1] https://www.khanacademy.org/

[2] https://www.coursera.org/

the world. Coursera is a for-profit company, but it also offers a number of free courses and provides financial aid to students who cannot afford to pay for a course or degree program.

On the other hand, various paying solutions have been developed. For example, Eedi[3] is an artificial intelligence-powered online tutoring platform developed by a UK-based company designed to support primary and secondary school students in their learning of mathematics and English.

1.1.4 A thesis between academic and industrial interests

My Ph.D. thesis has been situated in the context of an industrial-academic collaboration that started in January 2020. The collaboration was established between Sorbonne Université and Kartable through a Cifre contract. Kartable[4] is a French company that provides online educational resources, including interactive exercises, for primary, secondary, and higher education students. Its platform covers a wide range of subjects, including mathematics, physics, chemistry, biology, history, geography, and languages. Kartable also offers tools to track student progress. The company was founded in 2013 and has since become one of the leading providers of online education in France. However, in January 2022, Kartable could no longer support my contract, and the collaboration ceased. Despite this setback, I have continued my Ph.D. studies through an academic contract.

During my time at Kartable, I worked as an instructional engineer, which involved developing educational materials for students. Specifically, I was responsible for coming up with a set of exercises that were designed to meet the learning objectives and fit the knowledge components description. Therefore, the instructional engineering of resources in Kartable has been made following the decomposition of domain knowledge into knowledge components, which will be detailed in Section 2.1.1. Kartable interactive exercises are supposed to assess a specific Knowledge Component (KC), and this will be helpful in our experiments.

1.2 Motivations for modeling students

ITS rely on student modeling for providing functionalities such as knowledge state diagnosis or performance prediction. Student modeling is also a crucial component of EDM and LAK research works. It aims to represent the learner's knowledge and other meta-cognitive characteristics through variables and parameters. The student can be modeled directly in the TEL system or from the ITS data afterward. We present the motivations for modeling students as complex systems. In particular, we present the Knowledge Representation and Reasoning (KRR) and data-driven perspectives for complex system modeling. We also discuss how mixing these techniques may enhance the interpretability of student modeling.

[3] https://www.eedi.com/

[4] https://www.kartable.fr/

1.2.1 Modeling the learner as a complex system

A model is a simplified system representation used to explain, predict, or control the system's behavior. Models rely on features selected to be relevant regarding the focus on the modeled system. Modeling students is a complex task. We indeed consider learners as dynamic systems that can exhibit a wide range of behaviors and characteristics. Student modeling then involves studying multiple interconnected components and the processes between each other and the environment. To effectively model the learner, one must consider all of these components and their interplay, which can be challenging due to their nonlinear and dynamic nature. Student modeling can be seen as a complex system modeling, and modeling complex systems is a challenging but incentive task.

Exact mathematical approaches to complex system modeling are difficult to establish. The outcome of a complex system is the result of all the elements of the system and all the interactions between them. This requires the study of both the microscopic and macroscopic scopes of the system. The microscopic scope involves studying the local interactions of the components, while the macroscopic scope involves considering the whole system and the patterns that emerge from the microscopic interactions. These interactions reveal dependence between the variables that represent the system. Consequently, the mathematical formulation of whole complex systems often does not have analytical solutions.

> "All models are wrong, but some are useful."
> — George Box

This quote reveals how the comprehension of the properties of the complex system should guide the modeling even on intractable systems. To be relevant, a model must question the critical elements of system knowledge. Understanding its complexity would lead to more insightful modeling. Mathematical formulations of complex systems' macroscopic behavior, such as power-law statistical distributions, are often used to characterize higher-scale or aggregate output patterns. Still, they miss modeling all the microscopic elements representing the knowledge and their interactions. Our motivation for modeling students is to provide a learner model that highlights some of the learner's characteristics of interest in the educational context.

1.2.2 The KRR perspective

Knowledge Representation and Reasoning (KRR) was one of the first perspectives for modeling complex systems such as students [BL04]. Knowledge representation is the process of encoding knowledge in a structured and meaningful way so that computer systems can understand and process it. Broadly, it corresponds to modeling knowledge with the perspective of reasoning. It has been one of the former Artificial Intelligence (AI) research subfields [BL04], as it emphasizes the techniques to represent information about systems to provide the capability for computers to reason on it. Modeling knowledge and reasoning on

it requires mobilizing mathematical, statistical, or computational techniques.

Definition 1.2.1 (Knowledge representation and reasoning)**.** *Knowledge representation and reasoning refers to the technical problem of encoding human knowledge and reasoning into a symbolic language that enables it to be processed by information systems.*

In the context of a system, knowledge refers to the understanding and information experts have about the system, including both explicit and implicit knowledge and their beliefs about the system. Explicit knowledge emphasizes the facts or the information on the system that experts can write down. Implicit knowledge relates to knowledge that experts acquire through experience. Finally, beliefs about the system are the personal interpretation experts may have of the system.

This decomposition gives insights into the structure of KRR models. The system's knowledge can be broken into components, including concepts or relationships. Its representation should also consider whether elements of knowledge are associated with explicit knowledge, implicit knowledge, or beliefs, as it may influence the model. For instance, uncertainty refers to the degree to which the knowledge represented is uncertain or incomplete and is generally integrated into the knowledge representation. It will be studied further in Section 3.1.1. Knowledge representation is based on decomposing knowledge concepts or their attributes into variables. By breaking down knowledge into these components, we can use various techniques and tools, such as domain ontologies, semantic networks, or conceptual graphs, to represent the knowledge in a structured and meaningful way. This allows for effective reasoning and decision-making based on knowledge.

1.2.3 The data-driven machine learning perspective

The growing amount of data nowadays has brought the modeling of complex systems to new horizons. In numerous complex systems, data entries representing the target distribution are available [AT19]. Moreover, getting a large amount of data from a complex system may be easier than getting human expertise. Consequently, the interest in machine learning has massively increased these last few years, in particular, thanks to the increasing computational power of computers and the prominence of data in everyday life [PM15]. Data-driven machine learning allows us to explore connections between hidden system components. We lay the definition from the Cambridge dictionary down in Definition 1.2.2.

Definition 1.2.2 (Machine learning)**.** *Machine learning is the process of computers changing how they carry out tasks by learning from new data without a human being needing to give instructions in the form of a program [Dic].*

The goal of data-driven machine learning is for computers to learn from the data provided to carry out certain tasks. One understands that there are plenty of approaches to machine learning, as the definition of machine learning is wide. Performing machine learning consists in determining the instance of a given model family that fits the best to the available data.

We call this task "training the model" on training data. After training, processing other data with the trained model to make predictions is possible. The type of model varies depending on the target machine-learning task. In our case, the model is the knowledge representation.

1.2.4 Mixing KRR and machine learning approaches for student modeling

This work will highlight the interest in combining KRR and machine learning techniques for student modeling. In particular, we shed light on the gain of interpretability implied by this mix.

The most natural approach is to apply machine learning techniques to KRR methods. It determines the quantitative values of the knowledge representation from data, while KRR approaches define its qualitative structure. In a nutshell, it enhances the knowledge representation with information gathered from data. Such models are based on a formal mathematical and philosophical basis, relying on a KRR structure. They enhance the symbolic reasoning of KRR with data-driven methods. The KRR structure kept in machine learning applied to KRR provides the needed interpretability on the values learned from data. Formal mathematical and philosophical background ensures that each learned value relates to a tangible phenomenon described in the knowledge representation.

Definition 1.2.3 (Interpretable machine learning)**.** *Interpretable machine learning focuses on designing inherently interpretable models, i.e., presented in understandable terms to a human.*

On the other hand, another point of view on the mix between knowledge representation and machine learning is to apply an overlayer of knowledge representation to the output of machine learning techniques. Some machine learning techniques are too opaque, implying several issues, particularly about the confidence of humans interacting with the model (whether they are designers or users). KRR can be used to explain the decisions taken by these machine-learning black boxes. The overlayer provided by the inclusion of knowledge representation in machine learning techniques offers explainability on the algorithm decisions. A field of research called eXplainable Artificial Intelligence (xAI) has emerged from the wish to get explainable results from black-boxes algorithms [DBH18], which were uniquely result-oriented in the first place.

Definition 1.2.4 (Explainable machine learning)**.** *Explainable machine learning tries to provide post hoc explanations for existing black-box models, which are incomprehensible to humans.*

One of the key challenges in TEL is to develop interpretable student models that can provide teachers and learners with valuable feedback. Specifically, interpretable student models must rely on the modeling of concrete components of the complex system and tangible interactions between them. Developing interpretable student models can provide insights into how students learn, what they know, and what they don't know. Moreover, such interpretable

models can help teachers and learners make relevant decisions about the learning process, which can lead to improved student learning outcomes. Therefore, we concentrate our research on the techniques that apply machine learning to KRR methods.

1.3 Main contributions

Our work has resulted in several contributions. We introduce a framework for interpretable student modeling and technical improvements of existing machine learning techniques.

1.3.1 Developing a framework for interpretable student models

The main goal of this thesis work is to design and evaluate a framework for student modeling called E-PRISM, for Embedding Prerequisite Relationships In Student Modeling. The framework is designed to be adapted to a wide range of ITS.

E-PRISM aims to provide interpretable insights into the learning process and the prerequisite structure between the elements of domain knowledge. It uses dynamic Bayesian networks to represent domain and learner knowledge through an interpretable learner model. It models students' knowledge with dynamic knowledge states and considers the causal effect of the prerequisite relationships between knowledge components over time.

The E-PRISM framework relies on a combination of KRR and machine learning techniques, namely Bayesian networks with the clause of Independence of Causal Influences (ICI). ICI models are KRR approaches that introduce additional assumptions on the modeled system [DD06]. We show that using ICI models implies reducing the number of model parameters in the learner model to benefit model parameters' interpretability.

1.3.2 Providing metrics for discovering the prerequisite KC structure

The main contribution of this paper is to propose a method for identifying prerequisite relationships within a domain model from the study of the parameters of an interpretable learner model. The E-PRISM framework can be used to analyze learners' data and detect the prerequisite structure of the domain model.

We will examine how performing parameter learning with E-PRISM can provide insights into the learner's learning process and the prerequisite relationships between knowledge components. The E-PRISM framework has notably the potential to improve the learning experience by utilizing these prerequisite relationships in the design of personalized learning paths for students. We will also present an experimental study to illustrate the application of our proposed method in a real-world setting. Our proposed method is evaluated using real-world educational data, and the results are discussed in detail.

Finally, we define new metrics for assessing the causal impact of prerequisite relationships utilizing the interpretable parameters of E-PRISM. We use the proposed model to infer the

underlying prerequisite structure of a domain model from a dataset of real-world learner traces.

1.3.3 Developing a new approximate inference method for ICI-based BNs

The inference problem in Bayesian Networks (BNs) is computationally challenging, especially when the available data is extremely scarce. The E-PRISM framework exhibits convergence issues for the inference task because it relies on Bayesian networks with ICI-model Conditional Probability Distributions (CPDs) [DD06]. Nevertheless, inference is crucial for ITS application purposes, such as knowledge state diagnosis or performance prediction. We present a new stochastic technique for approximate inference in ICI-based BNs. Our approach allows making inferences from scarce data avoiding the stochastic potential well induced by the use of usual Monte-Carlo Markov chains techniques in the presence of deterministic relationships. Our method is advantageous in applications where interpretability is key, which is the case in education.

1.4 Outline of the manuscript

First, Chapter 2 covers how students are modeled in TEL systems and, more specifically, in ITS. It gives an overview of the techniques for updating the learner model, particularly the algorithms for predicting the learner's performance and the methods that search for the knowledge component structure of the domain model.

Chapter 3 covers Bayesian networks and their use as a model for knowledge-enhanced machine learning. It explains how uncertainty is handled, particularly how the probability theory can be used to reason under uncertain knowledge representation. It also provides an extensive overview of the inference and parameter learning techniques in Bayesian networks.

Chapter 4 details the E-PRISM framework. Specifically, it encompasses its architecture and its components. It deeply studies the learner model, defined as a new complex system model based on Bayesian networks with ICI-model CPDs.

Chapter 5 explains the techniques employed to update the learner model of the E-PRISM framework. After analyzing the complications of the usual inference techniques, it introduces novel approaches for approximate inference and parameter learning of ICI-based BNs. It details how it is applicated in the E-PRISM framework.

Finally, Chapter 6 covers the applications of the E-PRISM framework. It starts with comparing E-PRISM with other learners' performance prediction algorithms. Then, it studies insights into the prerequisite structure of the domain knowledge gathered from the learned E-PRISM parameters. It introduces new metrics to measure the existence and strength of a prerequisite relationship from the perspective of discovering the prerequisite structure expressed in the data.

Chapter 2

Student modeling in ITS

Learning environments that support and enhance education and learning thanks to digital technologies can be grouped under the umbrella term of Technology-Enhanced Learning (TEL) environments. This chapter aims to provide a comprehensive overview of particular TEL environments, Intelligent Tutoring Systems (ITS). ITS are computer-based educational systems designed to assess learners' knowledge and skills. They use artificial intelligence and instructional engineering principles to provide personalized and adaptive instruction to learners.

Definition 2.0.1 (Intelligent Tutoring System)**.** *An Intelligent Tutoring System (ITS) is a computer-based system that uses artificial intelligence and educational software to provide personalized and adaptive instruction to learners. ITS are typically based on various components to assess a learner's knowledge and skills, and then provide appropriate instruction based on the learner's needs.*

ITS can be used in a variety of settings, including schools, universities, and workplace training programs, and through a variety of platforms, such as software programs, mobile apps, or virtual assistants. They aim to simulate the experience of working with a human tutor, but with the added benefits of potentially being available anytime and being able to process and analyze large amounts of data. Overall, the goal of ITS is to provide learners with a tailored and efficient learning experience that is optimized for their individual needs.

According to Wenger, the general structure of ITS is composed of four different models, that are, the domain, learner, pedagogical, and communication models [Wen86]. After quickly describing the role of each of these models, we give a closer look at learner modeling. Woolf highlights three issues with student models in ITS: representing student knowledge, updating student knowledge, and improving tutor performance [Woo10]. In this chapter, we focus on the state-of-the-art for tackling these research questions. First, we review the different representations of student knowledge in learner modeling. Then, we present the techniques for updating the student knowledge from data. Finally, we tackle the issue of improving the performance of the ITS. Specifically, we focus on the search for the prerequisite KC structure.

2.1 The central role of the learner model

Learner models are a crucial component of ITS. They represent the learner's current knowledge, skills, and learning preferences, and they serve several purposes within an ITS. Learner models are employed to adapt the instruction to the learner's needs and track their progress over time.

In this section, we introduce learner models in the structure of ITS. We then provide an overview of the techniques for updating the learner model, restricting the study to learner models focused on modeling the learner's knowledge. Finally, we present hybrid learner models that mix symbolic and data-driven approaches for learner modeling.

2.1.1 Components of an ITS

In order to support learning and reasoning about students' knowledge, an ITS must be able to represent and analyze the knowledge in question. Wenger describes the architecture of an ITS as a sum of four main components: the domain, learner, pedagogical, and communication models [Wen86].

Domain model

The domain model is a representation of the knowledge of the domain being taught by the ITS [NMB10]. It includes the concepts, relationships, and rules that define the subject matter. The domain model serves as a reference for the ITS to determine the appropriate instructional content to present to the learner and to assess its progress and understanding. It consists of expert knowledge, which can be structured and formalized using various techniques.

Definition 2.1.1 (Domain model)**.** *A domain model in an ITS is a representation of the domain knowledge being taught by the ITS, including the concepts, relationships, and rules that define the subject matter.*

A popular depiction of domain knowledge consists of a decomposition of the knowledge into Knowledge Components (KCs) [Pel17]. It has been introduced in the framework Knowledge-Learning-Instruction (KLI) [KCP12].

Definition 2.1.2 (Knowledge component)**.** *A knowledge component is "an acquired unit of cognitive function or structure that can be inferred from performance on a set of related tasks." [KCP12]*

Knowledge components are related to test items on which learners are evaluated. Koedinger et al. describe the temporal granularity of such items as "unitary tasks" lasting a dozen seconds [KCP12]. This temporal granularity may fluctuate, as do the names given to knowledge components (skills, concepts, techniques, ...) [Pel20]. The relationships between different concepts in the domain model are often represented with a hierarchy of concepts to show how they are connected and related to one another [DF12]. Ontologies are also often used

in education, as they provide a standardized representation of knowledge that can be used by different educational systems and facilitate the interoperability of educational resources [MG14, AYGA15].

We note that some authors include the domain model in the learner model [Pel17]. The domain model was indeed supposed to be fixed and defined by experts. However, with the increase in the amount of data on learner's transactions, more and more approaches have considered the influence of the learner on the domain model. Consequently, Bayesian Network (BN) approaches for modeling the domain knowledge have arisen [CGV02, CMPdlC+05, MLPDLC10, KKSG17]. The Conditional Probability Distributions (CPDs) defining the BN describing the domain model are generally elicited by experts or associated with a learner model to fit the data better.

Learner model

The learner model is a representation of the learner's knowledge, skills, and learning preferences. It may include information about the learner's prior knowledge, learning style, and progress through the instructional material. The learner model is continuously updated as the learner interacts with the ITS. It is used by the ITS to adapt the instruction to the learner's needs and to track their progress over time.

Definition 2.1.3 (Learner model)**.** *A learner model in an ITS is a representation of a student's knowledge, skills, and characteristics relevant to the specific subject of the ITS.*

Woolf differentiates two types of learner models, depending on their focus on learner knowledge [Woo10]. We note that different authors and researchers may have different ways of categorizing and describing learner models [DdB12, AKIB19, Pel17, CV13].

On the one hand, overlay modeling represents the learner's knowledge as a set of annotations or layers on top of the domain model. It considers the learner's knowledge a set of independent concepts or skills, with a separate score or proficiency level for each concept [CG77]. It overlays the learner's current state onto the domain model, representing the domain knowledge. The overlay learner model can be updated as the learner progresses through the learning material, allowing it to adapt to the learner's changing knowledge and skills. It generally uses a state-based representation to track the learner's knowledge and learning progress [ND08]. The learner model then consists of a series of states, and the transitions between states represent the acquisition or loss of knowledge. Still, an overlay model can use a different method of representing the learner's knowledge, such as probabilistic representation [BM07]. Subtypes of overlay models include stereotypes, which are pre-defined models that represent typical learners in a particular domain [Ric79]. Stereotype models are built by overlaying domain knowledge onto a predetermined set of characteristics associated with a particular group of learners, such as their age, gender, or cultural background. Also, we note that overlay models can sometimes include misconceptions on top of domain knowledge, such as

in perturbation models [Bur82].

On the other hand, error-catalog modeling involves representing the learner's knowledge as a collection of past experiences or cases that can be used to guide problem-solving. Error-catalog models aim to track and catalog the errors the learner makes during the learning process in bug libraries. The learner model then keeps a record of the errors made, as well as the context in which they occurred and any relevant information about the learner's knowledge and skills at the time [SCS00]. This information is used to identify patterns in the learner's errors and to provide feedback and guidance to help the learner overcome their mistakes and improve their learning. Contrary to perturbation models, which introduce "perturbation" to the learner's knowledge, error-catalog learner models focus on identifying the errors the learner makes while interacting with the learning material. Error-catalog models may also use state-based representation, but, in this case, the states represent the learner's errors rather than their knowledge.

Pedagogical and communication models

The pedagogical model specifies the instructional strategies and methods used by the ITS to present the content to the learner. It uses information from the domain model and learner model to make decisions about tutoring strategies and actions. This decision-making process is especially based on a diagnosis of the learner's behaviors, current knowledge, and personal characteristics (such as personality, motivation, learning style, and communication style). The pedagogical model determines the sequence and pacing of the instructional material, as well as the types of activities and feedback provided to the learner.

The communication model enables the ITS to communicate with the learner through a user interface. It determines the layout and design of the interface, as well as the tone used to present the content and communicate with the learner. It handles the presentation of tasks, problems, and information to the learner. It also gathers information about her activities to provide assistance in the form of advice or interface adaptation.

Pedagogical and communication models will not be studied further in this work. We assume the ITS in which our work will be implemented already relies on existing pedagogical and communication models.

2.1.2 Techniques for updating learner models

Whether overlay or error-catalog models, learner models rely on a set of variables modeling different aspects of the learner. We detail the characteristics usually studied in learner models and focus on the various techniques employed to update the learner model.

Modeled learner characteristics

Learner characteristics refer to the personal attributes and characteristics of the individual who is learning. These include knowledge state, personality, motivation, communication style, and more [DdB12]. González et al. differentiate the learner's characteristics as a function of their relationship with the domain model [GBL06]. In contrast, Jeremić et al. differentiate static characteristics, set before learning, and dynamic characteristics, modeled from learner traces and changing during learning sessions [JJG12]. These characteristics are taken into account when adapting instruction to the learner's needs and tracking progress over time. For example, if a learner has visual learning preferences, the ITS might use more visual aids in the instruction [CKP+06]. If a learner has a low level of motivation, the ITS might use more incentives to keep them engaged [VP98, DA15]. The ITS can provide a more personalized learning experience by considering learner characteristics.

The knowledge of the learner is one of the learner's characteristics. It is the most studied dynamic aspect of learner modeling, according to Desmarais et al. [DdB12]. The learner's knowledge refers to the concepts, procedures, and rules the learner has learned or is currently learning. In overlay learner modeling, the representation of the learner's knowledge is directly related to the chosen modeling of the domain: the learner model must represent the learner's knowledge in a way that is compatible with the domain model so that the ITS can reason about the learner's understanding and provide appropriate instruction.

Model-tracing techniques

Model tracing involves constructing a model of the learner's problem-solving process by tracking their actions and inferences as they solve a problem. In a nutshell, it "traces" the learners' learning process as they work through various tasks. Model-tracing techniques are based on psychological learning theories and use this knowledge to guide instruction [CV13]. In the former model-tracing techniques, each possible action from the learner and the resulting feedback were referenced. Cognitive tutors, the most well-known examples of model-tracing tutors [KAH+97] presented in Section 1.1.2, describe problem-solving as a set of production rules: the tutor follows the learner's reasoning by analyzing the applied rules. Knowledge tracing can be seen as a particular case of model tracing [Woo10]. Knowledge tracing considers the learning process from a macroscopic point of view. It can also be based on statistical approaches: for instance, deep knowledge tracing uses neural networks [PBH+15]. Overall, model-tracing techniques are often applied to overlay learner models.

Model-tracing techniques like knowledge tracing often use probabilistic representation to track the learner's knowledge and learning progress [Pel17]. The learner model then consists of a set of probabilities representing the likelihood that the learner has learned a particular concept or skill. Techniques for building these kinds of learner models include machine learning algorithms. In particular, cognitive diagnosis algorithms are a type of machine learning algorithm employed in learner models to track the mastery of underlying skills that

will be further detailed in Section 2.2.3.

Constraint-based techniques

Constraint-based modeling involves updating the learner's knowledge with a set of constraints that the learner must follow in order to demonstrate her understanding of the domain knowledge [Ohl94, Mit12]. This means that the learner's progress is represented by the satisfaction or violation of these constraints. The ITS can then adapt the instruction based on whether or not the learner is able to follow these constraints, which can be represented in various ways, such as using a set of logical rules [Mar99] or using a probabilistic graphical model [MM00]. Consequently, constraint-based modeling is a technique usually employed for error-catalog learner models, but it can also be applied to overlay models. Still, it requires massive efforts to be updated.

Example-tracing techniques

Example tracing involves constructing a model of the learner's problem-solving process by comparing their actions to a set of worked examples. They use the learner's actions to infer their understanding. They focus on examples and how the learner applies the domain knowledge to solve specific problems [AMSK09]. These techniques often involve the use of a set of rules or procedures that determine how new examples should be processed and integrated with the learner's existing knowledge. They are characterized by their capacity to store and retrieve examples and to analyze examples to identify important features and relationships. Example-tracing models then compare new examples to stored examples to identify similarities and differences, using them to generate explanations or predictions. While model-tracing techniques define a learner's problem-solving process to infer their current knowledge [Paq22] by understanding the underlying principles that govern problem-solving, example-tracing techniques are more concerned with using specific examples to guide the learner through new problems.

Model-tracing and example-tracing techniques may overlap. For instance, example-tracing techniques include machine learning algorithms, as they use many student examples to build a model of the student's knowledge and learning progress. These techniques typically involve training a machine learning algorithm on a dataset of student examples and then using this trained model to make predictions about a student's performance on future problems [WPB01]. The chosen machine learning algorithm can rely on cognitive theories [ACKP95] and, then, also be model-tracing.

2.1.3 Mixing symbolic and data-driven approaches in learner modeling

Among these techniques for updating the learner model, the majority can either be considered from a symbolic or a data-related point of view, and they can even be mixed in hybrid learner models.

Symbolic and data-driven learner models

Symbolic approaches to learner models rely on explicitly defined rules to represent the learner's knowledge. These models are often based on cognitive theories and can be more interpretable, but they may be less flexible and require more manual effort to design and maintain. These rules may be based on expert knowledge or experience and may be designed to adapt to the specific needs and characteristics of the learner. Constraint-based techniques use, most of the time, a symbolic approach.

On the other hand, data-driven approaches use statistical techniques and machine learning algorithms to learn models from data automatically. These models can be more flexible and efficient but may be less interpretable and require more data to learn accurately. In recent years, data-driven approaches, particularly machine learning algorithms, have gained popularity due to their ability to handle large amounts of data and adapt to changes in the learner's knowledge over time. Example-tracing techniques are, most of the time, data-driven.

Hybrid learner models

It is worth noting that some techniques for updating learner models may incorporate elements of both approaches [Pel17]. For example, a model-tracing technique may use a cognitive theory as a basis and incorporate data-driven techniques to learn and update the model [ACKP95, FVNN10].

When a learner model mobilizes symbolic and data-driven approaches, we call it a hybrid learner model [FVNN10]. It is worth noting that hybrid learning models are often used when the data available for training a data-driven model is limited or when the problem being solved requires a high level of human expertise. In these cases, symbolic techniques can provide a solid foundation for the model. At the same time, the data-driven system can learn from the data to improve the accuracy of the predictions or decisions made by the model.

Structure of hybrid learner models

The way symbolic and data-driven techniques interact with each other can vary. For example, the output of the data-driven system may be used to adjust the rules in the model, or symbolic techniques may be used to pre-process the input data before it is passed to the data-driven system. Pelánek represents the main components of learner models aiming at tracing learners' knowledge across multiple items and over time [Pel17]. His representation, depicted in Figure 2.1, particularly fits for portraying hybrid learner models, as it describes them into elements that can be seen as symbolic or data-driven components of the model.

The knowledge model in Figure 2.1 corresponds to the learner model depicted in Section 2.1.1. It is related to the domain model in the manner of overlay learner models, as it defines the current knowledge state of the learner regarding the domain model and, more

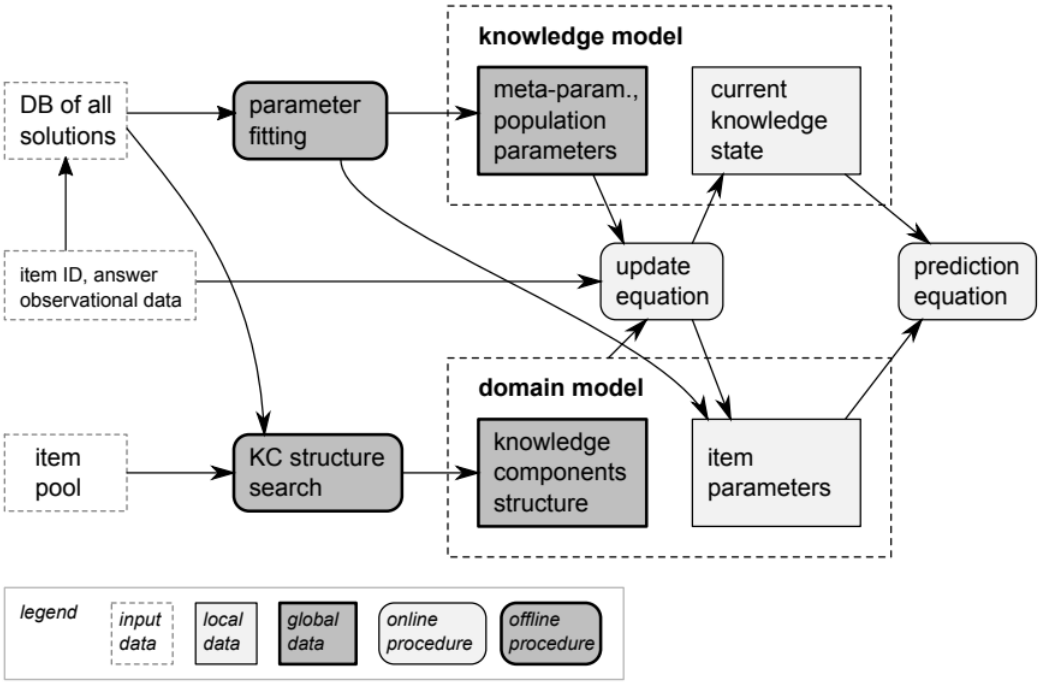


Figure 2.1: Representation of a learner model aiming at tracing learners' knowledge with a long-term perspective. This representation portrays greatly hybrid learner models. The figure is borrowed from Pelánek's work [Pel17].

particularly, the KC description. The depicted model is then an overlay model. Pelánek differs four procedures for updating the learner model.

On the one hand, the update and the prediction equations correspond to the techniques employed for updating the learner model. As reported previously, these two procedures can be symbolic or data-driven. The update equation is deeply related to the prediction equation, as it is generally one of its components. We detail the state-of-the-art approaches for predicting the learner's performance in Section 2.2.

On the other hand, the parameter fitting and the KC structure search procedures are exclusively data-driven. They result from the latest machine learning research and are not described in Woolf's work [Woo10]. The parameter fitting procedure is required to include data-driven approaches in learner models, as it allows us to fit the input data. KC structure search is surely the most computationally-costly task for hybrid learner models, and it is the main focus of our work. We focus more particularly on the prerequisite relationships that could exist between KCs. We detail the state-of-the-art approaches for searching the prerequisite KC structure of the domain model in Section 2.3.

2.2 Learners' performance prediction

Predicting the learner's performance can be helpful in a variety of ways, both in the construction of learner models and in their use. It can help to identify which features or variables are most important in predicting future performance. It can also help to tailor instruction to the needs of individual learners, allowing for more personalized and compelling learning experiences.

In this section, we first study the various purposes of the learners' performance prediction. In particular, we expose how the prediction goal may change depending on the type of learner model. Then, we focus on two kinds of learners' performance prediction algorithms: logistic regression models and cognitive diagnosis algorithms.

2.2.1 Purposes of learners' performance prediction

Learners' performance prediction algorithms are central to learner modeling because they aim to predict students' performance on assessments, diagnose their strengths and weaknesses, and track their learning progress over time.

Updating and evaluating the learner model

The straightforward role of learners' performance prediction algorithms is updating and evaluating learner models. These algorithms use past performance data to make predictions about learners' future performance on a particular task or set of tasks. These predictions can then be used to update the learner model with new information about learners' knowledge and skills. Tracking the learners' knowledge states is the main purpose of learner models. Consequently, learners' performance prediction algorithms are essential in learner models.

Also, by comparing the predictions made by the learner model with the actual learners' performance, it is possible to determine the level of fit between the model and the real-world data. This information can be used to refine and improve the learner model over time, as machine learning algorithms require predictions for model learning. This point is specific to data-driven learner models.

In addition to updating the learner model, the learners' performance prediction algorithms can also be used to assess the performance of learner models and compare them with each other. One can define metrics for measuring how well the model can predict a learner's behavior, such as the Area Under the ROC Curve (AUC) or the Root Mean Square Error (RMSE) – see Section 5.2.4 for further details on the available metrics. This can be used to indicate which learner model is the most accurate or effective regarding these metrics.

Obtaining insights into learning

The better the prediction, the fitter the parameters. Indeed, most learner models rely on a set of parameters, which are updated at the same time as the learners' performance prediction. Depending on the kind of the studied learner model, these parameters can be associated with features of interest, such as in example-tracing models, or with cognitive phenomena, such as in model-tracing models. Consequently, for some learner models, correctly predicting the learner's performance means the underlying parameters give meaningful information about the learner.

Still, the interpretability of parameters in learner models differs from one model to another. Model-tracing learner models have been designed to provide more insightful parameters. On the contrary, example-tracing learner models aim to be the best predictive learner model, and the parameter interpretability of such models may not be significant. Therefore, if they can help identify the most important features in future performance prediction, no predominant insights may be extractable from the model structure.

Adapting the instruction

Finally, predicting the learner's performance is one of the most valuable tasks of learner models because it can help educators better understand students' needs and abilities and tailor their teaching and learning approaches accordingly. By predicting how students are likely to perform on assessments, educators can identify areas where students may be struggling and provide targeted interventions to help them improve. This can be particularly useful when students struggle to keep up with the rest of the class, as it allows educators to provide additional support and resources to help the students catch up.

Moreover, predicting student performance can also be fruitful for monitoring the effectiveness of different teaching and learning approaches. By comparing students' actual performance to the predictions made by the learner model, educators can assess the effectiveness of different teaching strategies and identify approaches that are more or less successful at helping students learn.

2.2.2 Logistic regression algorithms

Logistic regression is a popular and widely-used method for learners' performance prediction in educational settings.

Principles of logistic regression

Logistic regression is a statistical method for predicting the probability of an event occurring, given certain predictors [KDG+02]. It is used to model a binary outcome, such as whether a student will succeed in a test item. Indeed, logistic regression can be used to predict the likelihood of a student correctly answering a question or completing a task based on a set of

independent variables, such as their prior knowledge or their performance on related tasks. Logistic regression uses a feature vector defined manually from learner traces' data, and it applies a logistic function to the chosen set of features.

Definition 2.2.1 (Logistic regression)**.** *Logistic regression is the task of estimating the parameters of a logistic model, which models the probability distribution of a variable Y as a logistic function, expressed in Equation 2.1, of a linear combination of one or more independent variables* $\mathbf{Z}$*.*

$$\forall z \in \mathbb{R}, \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.1}$$

Mathematically, logistic regression is a generalized linear model that uses the logistic function to model the probability of an event occurring as a function of one or more predictor variables. The coefficients of the predictor variables in the logistic regression model can be estimated using maximum likelihood estimation.

Example 2.2.1 (Logistic regression)**.** *Suppose we model the probability of passing an exam (which has binary outcomes) with a logistic regression model. It predicts the probability of a student passing an exam based on their study habits and test-taking ability, following Equation 2.2*

$$y = \frac{1}{1 + \exp(-b_0 - b_1 x_1 - b_2 x_2)} \tag{2.2}$$

where y is the probability of passing the exam, x_1 is the variable representing study habits, x_2 is the variable for the test-taking ability, and b_0, b_1, and b_2 are the model coefficients. b_0, b_1, and b_2 are estimated using maximum likelihood estimation.

Logistic regression relies on discriminative training. The only relationship of interest is indeed the conditional probability distribution $P(Y \mid \mathbf{Z})$. From well-designed features, logistic regression models define the outcome as a function of the features and only study this conditional relationship.

With binary variables, one uses one-hot encoding to convert the variables into a format adapted to logistic regression models for predicting the probability of the target variable. When we one-hot encode a categorical variable, a new binary column is created for each unique category of the variable. Each row is then marked with a 1 in the column corresponding to its category and a 0 in all other new columns. The parameters of the logistic regression model are the coefficients associated with each column, representing the log-odds of the outcome for the category they are associated with.

Item response theory

Item Response Theory (IRT) is a statistical framework used to model the relationship between individuals' ability and their responses to test items. It supposes that the answers of the learner on test items are modeled by one or more hidden variables [HSR91]. The most

well-known IRT model is the Rasch model, which assumes that the probability of a correct response to a test item is a logistic function of the person's ability and the item's difficulty. It is expressed with Equation 2.3.

$$P(Y_{i,j} = 1) = \sigma(\alpha_i - \delta_j) \tag{2.3}$$

The Rasch model considers the knowledge of a learner i as a unique variable α_i. The probability of answering correctly is computed by comparing the learner's knowledge α_i with δ_j supposed to represent the difficulty of the test item j. The main advantage of IRT is that it allows for measuring abilities on a continuous scale rather than just categorizing individuals into fixed ability groups. It is also possible to estimate the difficulty of test items and the ability of individuals even with a small number of items. IRT models are widely used in educational testing and psychological assessment.

In addition to the traditional unidimensional IRT, Multidimensional Item Response Theory (MIRT) can simultaneously measure multiple abilities. These models are helpful in situations where there are multiple knowledge components are being assessed at the same time. The equation ruling MIRT is depicted in Equation 2.4.

$$P(Y_{i,j} = 1) = \sigma(\boldsymbol{\alpha}_i \cdot \boldsymbol{d}_j - \boldsymbol{\delta}_j) \tag{2.4}$$

MIRT supposes that the knowledge $\boldsymbol{\alpha}_i$ of the learner i is a vector rather than a scalar. The characteristic vector of the test item $\boldsymbol{d}_j$ indicates the knowledge components related to the item j, and $\boldsymbol{\delta}_j$ is its difficulty vector [Rec09].

Other logistic regression algorithms

IRT and MIRT models are logistic regression models. They use simple features, but other logistic regression models use engineered and more complex features. Other noticeable logistic regression algorithms integrate new features and, therefore, new parameters. Learning Factor Analysis (LFA) [CKJ06] is represented by Equation 2.5.

$$P(Y_{i,j} = 1) = \sigma\left(\alpha_i + \sum_{k \in KC(j)} (\beta_k + \gamma_k N_{ik})\right) \tag{2.5}$$

In LFA, the learner parameters are similar to those of IRT. α_i represents the ability of the learner i. β_k is the bias for the KC k, and γ_k is the bias for each opportunity of mastering the KC k. $N_{i,k}$ is an attempt count feature: it is a categorical variable that represents the number of attempts of the learner i on the KC k.

Performance Factor Analysis (PFA) [PCK09] introduces the success of the attempt as a

feature in the model. It can be expressed with Equation 2.6.

$$P(Y_{i,j} = 1) = \sigma \left(\sum_{k \in KC(j)} \beta_k + \gamma_k S_{i,k} + \delta_k F_{i,k} \right) \tag{2.6}$$

In PFA, β_k is also the bias for the KC k. γ_k (resp. δ_k) is the bias for each opportunity of learning the KC k after a successful (resp. unsuccessful) attempt. $S_{i,k}$ and $F_{i,k}$ represent the number of successes and failures of the learner i over test items that requires the KC k. They are count features represented by categorical variables.

DAS3H [CPBV19] is an extension of DASH [LSPM14]. It integrates the concept of temporal windows by computing count features about it. It is described with Equation 2.7.

$$P(Y_{i,j,t} = 1) = \sigma \Big(\alpha_i - \delta_j + h_\theta \left(t_{i,j,[1,l]}, y_{i,j,[1,l-1]} \right) \Big) \tag{2.7}$$

Such as IRT, DAS3H considers α_i and δ_j for representing the ability of the learner i and the difficulty of the test item j. Also, similarly to PFA, it integrates a parameter for the bias associated with the KC k. These parameters are integrated into the function h_θ that processes the temporal distribution t and the outcomes of past practice y.

Finally, the Best-LR [GKS+20] algorithm is depicted in Equation 2.8.

$$P(Y_{i,j,t} = 1) = \sigma \left(\alpha_i - \delta_j + \phi(s_i) + \phi(f_i) + \sum_{k \in KC(j)} (\beta_k + \gamma_k S_{i,k} + \delta_k F_{i,k}) \right) \tag{2.8}$$

Best-LR mainly consists of combined IRT and PFA features. Its parameters indeed encompass the ability of the learner, the difficulty of the test item, the bias associated with the KC, and the parameters associated with the counts of successes and failures on KCs. It also includes parameters associated with a count feature of successes $\phi(s_i)$ and failures $\phi(f_i)$ uniquely related to the learner i.

2.2.3 Cognitive diagnosis algorithms

Cognitive diagnosis algorithms are other kinds of learners' performance prediction algorithms. They model the learner's knowledge state to predict her answers.

About cognitive diagnosis

The cognitive diagnosis is the process of inferring the learner's knowledge state to identify her learning difficulties [TH06].

Definition 2.2.2 (Cognitive diagnosis algorithm)**.** *A cognitive diagnosis algorithm is a type of algorithm that model the underlying cognitive processes or knowledge states to understand*

the learner's transactions with an ITS.

Cognitive diagnosis algorithms can be perceived as learners' performance prediction algorithms. Still, it is important to note that learners' performance prediction and cognitive diagnosis algorithms are different. While both may be used to predict learner performance, cognitive diagnosis algorithms are specifically designed to identify the underlying knowledge or skills that a learner has or lacks. In contrast, learners' performance prediction algorithms may consider a wider range of variables for making their predictions.

Non-temporal Bayesian network approaches

Some cognitive diagnosis models can be represented by Bayesian Networks (BNs). BNs will be extensively discussed in Chapter 3. They are used to compute the probability of answering correctly by modeling the learner's mastery of knowledge components.

Psychometric models such as Deterministic Input Noisy And (DINA) [Hae89] or Noisy Input Deterministic And (NIDA) [JS01] models introduce auxiliary variables to do so. DINA and NIDA models are two types of constraint-based learner models. DINA models assume that a learner has a set of latent ability variables that determine whether they will be able to answer a particular problem correctly or not. They link the variable representing the correctness of a learner's answer on a test item with the variables representing the mastery of the required knowledge components. DINA model considers that the success on a test item is a AND function of the mastery of the needed knowledge components with noise. NIDA introduces noise on the variables that represent the mastery of the prerequisite knowledge components to compute the probability of answering correctly to a test item. The structure of the NIDA model is a kind of Independence of Causal Influences (ICI) model which suggests that deterministic relationships are not perfect in the real world and introduce noise sources in it [DD06], see Section 3.1.2 for more details on ICI models. Other deterministic functions may also be applied to the KC mastery variables to perform cognitive diagnosis models with BNs. For instance, Deterministic Input Noisy Or (DINO) [TH06] uses the OR function.

Other non-temporal Bayesian networks can be used to infer the learner's knowledge. Millàn et al. and Carmona et al. use the relationships stored in their domain model [CMPdlC+05, MLPDLC10] to infer the probability of correctly answering. Variables associated with misconceptions and relationships with more general concepts are introduced in the network to predict the success of the learner on a test item. Desmarais et al. also use a Bayesian network to predict the correctness of learners' answers [DdB12]. They consider prerequisite relationships between test items to enhance the prediction. Moreover, they propose a set of rules to extract these prerequisite relationships from a statistical study of the learner traces. We will detail this perspective in Section 2.3.

In these Bayesian networks, the state of knowledge is supposed to be static: the probability of mastery will only evolve as a function of the evidence of the learner's answers. The

variables representing the mastery of knowledge components are time-independent. Their value is only updated when evidence on the Bayesian network is updated. The evidence itself is time-independent: the learner traces are considered to give the same information whenever they are collected.

Bayesian knowledge tracing

Knowledge tracing is a category of model-tracing learner models. It supposes the knowledge of the learner is dynamic and evolves over time. The learner's knowledge state is defined as a function of time.

The most famous knowledge tracing algorithm is Bayesian Knowledge Tracing (BKT). BKT has been introduced by Corbett and Anderson [CA94]. It relies on a hidden Markov model. The learner's knowledge is modeled for each knowledge component by a binary random variable representing the learner's mastery of that particular knowledge component. BKT can be represented as a dynamic Bayesian network that models the learner mastery of a knowledge component over time, as shown in Figure 2.2.

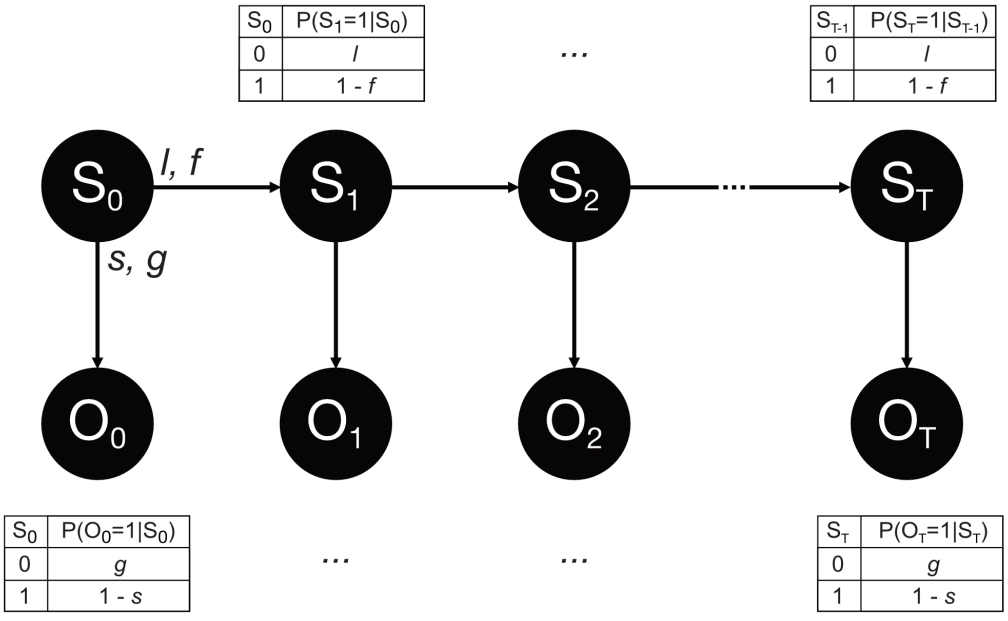


Figure 2.2: Unrolled dynamic Bayesian network representation of BKT. In this representation, we integrated the forget parameter f that is not considered in classic BKT. It represents the probability for the learner to forget the mastery of the corresponding KC between two timeslices.

It infers the probability for the learner to answer correctly, thanks to multiple parameters attached to tangible phenomena. BKT relies on the l, s, and g parameters. The "learn" parameter l relates the learner's probability of learning between two timeslices. The "slip" parameter s is the probability of answering badly while the knowledge component is mastered. Lastly, the "guess" parameter g describes the probability of answering correctly while the

knowledge component is not mastered.

Several variants of BKT have been developed over the years. Each introduces new specifications for the model parameters and complexifies the structure of the latent Bayesian network [dBCA08, PH10, PH11, QQL+11, YKG13, KLM16]. For instance, Yudelson et al. consider the "slip" and the "guess" parameters dependent on both the learner and the test item [YKG13], while Pardos et al. suppose the "learn" parameter to vary depending on the KC [PH11]. The Python package `pyBKT` offers an efficient and easy-to-use implementation of the several variants of BKT [BWP21].

While BKT and its variants only care about one KC at a time, Käser et al. use a Dynamic Bayesian Network (DBN) to model the learner knowledge and consider all the knowledge components of the domain knowledge simultaneously [KKSG14]. If this work has been categorized as a variant of BKT [SWHM22], it introduces new variables compared to BKT to represent the learner's mastery of every knowledge component. It also considers prerequisite relationships at each timestep of the Bayesian network with arcs between KC mastery variables.

Consequently, the complexity of such a Bayesian network is much higher than any variant of BKT. The inference – i.e., the computation of the joint distribution given some evidence on the knowledge state – of such a complex Bayesian network might not even be tractable [Pel20]. The number of parameters is dependent on the size of prerequisite relationships. Viable results may be hard to compute in large Bayesian networks with numerous arcs, according to Pelánek [Pel20]. For instance, training BKT+, a variant of BKT, takes several days, while training logistic regression or classic BKT takes a few minutes [KLM16]. Approximation techniques can be used to tackle the tractability of the model. For example, Käser et al. approximate the DBN with a log-linear model and use duality to learn the model [SHPU12]. Other techniques, such as variational inference [WJ+08] and loopy belief propagation [MWJ99], can be employed. We refer to Chapter 3 for further details on inference techniques in BNs.

Deep knowledge tracing

Deep learning techniques have arrived into learner modeling and have recently gained popularity [SWHM22, Pel17]. These techniques are based on neural networks. One of the main advantages of deep learning techniques is that they can automatically learn and extract features from large and complex datasets like logs of student responses to multiple-choice questions. In consequence, deep learning models do not require human feature engineering but need a large amount of data to be efficient [KLM16].

Definition 2.2.3 (Neural network)**.** *A neural network is a model composed of three sets of variables linked with each other:*

- *the visible set V, which describes the variables associated with inputs;*

- *the hidden set H, which is potentially decomposed in several layers;*
- *the output set O, which describes the variables associated with outputs.*

At a high range, neural networks aim to study the deep features of inputs and the structure between them, thanks to their hidden layers.

Example 2.2.2 (Neural network)**.** *Let's use neural networks rather than logistic regression to introduce an example similar to the Example 2.2.*

The neural network's input layer would have three neurons, one for each input variable: previous test scores, attendance record, and study habits. The output layer would have one neuron that represents the final exam performance. The input layer would pass the values of the three input variables to the hidden layer, composed of several neurons. Each neuron would apply a non-linear transformation to the input values, allowing the network to learn complex relationships between the input and output variables. The values from the hidden layer would then pass to the output layer, where they would be used to predict the final exam performance.

The network is trained using a dataset of students' previous test scores, attendance records, study habits, and final exam performance. The network adjusts the values of the weights between the layers to minimize the error between the predicted and actual final exam performance. Once the network is trained, it can be used to make predictions about students' performance based on their previous test scores, attendance record, and study habits.

Deep Knowledge Tracing (DKT) [PBH+15] is a deep learning model for the knowledge tracing task. DKT uses a neural network to learn a non-linear model of the learner's knowledge. This allows it to capture more complex patterns in the data and make more accurate predictions about a learner's performance. Numerous variants of DKT have been developed since then, but Schmucker et al. only relate minor performance gains relative to DKT [SWHM22], except for Self-Attentive Knowledge Tracing (SAKT) [PK19].

DKT and its variants are considered to be cognitive diagnosis models because they aim to infer learners' knowledge states with the hidden layers of the network to predict their answers. However, it is important to note that not all deep learning models for learner modeling are cognitive diagnosis models. Some models may be designed for other purposes, such as predicting students' performance or engagement, and may be designed to identify patterns in the data rather than inferring students' knowledge state [CV13].

2.2.4 On the interpretability of learners' performance prediction algorithms

We have presented in Sections 2.2.2 and 2.2.3 several families of algorithms for predicting the learner's performance. Whether they are based on engineered features or modeled cognitive phenomena, we analyze and compare the interpretability of their parameters.

Interpretability in logistic regression models

Logistic regression models are relatively easy to appreciate. Their results can be straightforwardly understood, as the model estimates the effect of each predictor on the outcome. Nevertheless, the interpretability of the parameters issued from logistic regression models is hard to apprehend. The parameters of these algorithms are computed from a one-hot encoder of the data. Consequently, the logistic nature of the algorithm implies multiplicative interpretability of the parameters [Mol22], and logistic regression is thus only locally interpretable.

Example 2.2.3. *Let's consider the logistic regression model in Equation 2.9.*

$$P(Y = 1 \mid \mathbf{X}) = \sigma\left(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n\right) \tag{2.9}$$

Changing the value of a feature X_j in the logistic regression model in Equation 2.9 changes the odds for $Y = 1$ multiplicatively by e^{β_j}, holding all other covariates constant.

Interpretability in models based on Bayesian networks

The non-temporal Bayesian networks model the knowledge of the learner with random variables. The parameters of these models are conditional probabilities between the variables corresponding to the modeled phenomena. They are then interpretable as they directly relate the probability of a causal effect [MBL21, Mol22]. However, similarly to logistic regression models, the non-temporal Bayesian networks do not consider the dynamicity of the learning process.

On the contrary, in Bayesian Knowledge Tracing, variables represent the mastery of a given knowledge component over time. Cognitive phenomena, such as learning and forgetting, are directly represented in the model, and some parameters are related to them. The interpretability of the model parameters benefits from the general interpretability of Bayesian networks [MBL21]. They also relate to tangible conditional probabilities.

Nevertheless, using an approximation for model learning can cause some trouble with the interpretability of the learned parameters. Käser et al. indeed note that the interpretability of the parameters is not ensured because the learning of the model only implies the convergence to a local optimum [KKSG17]. Also, a high number of parameters in the Bayesian network may also result in a more difficult model interpretation, as the range of the events represented in CPDs can become very narrow.

Interpretability in deep learning models

Finally, if deep learning models claim to outperform other learners' performance prediction algorithms on the predicting task, this advantage has a serious cost on the interpretability of the model. For instance, Deep Knowledge Tracing has tens of thousands of parameters

that are nearly impossible to interpret [KLM16]. Also, DKT learns an ability model rather than a skill mastery model [DL21]. The variables encoded in the model indeed represent the learners' ability to answer questions rather than their KC masteries.

2.3 Discovering the prerequisite KC structure

Some of the algorithms presented in Section 2.2 use the prerequisite structure of the domain model for predicting the learner's performance. Other models recover the prerequisite relationships thanks to predictions. Retrieving the prerequisite structure of the domain model is a challenging task in the field of Learning Analytics and Knowledge, as it provides insights into how the different Knowledge Components (KCs) are related and how they build upon each other. This information can be used to personalize the learning path for the students by recommending them learning activities that match their current state of mastery.

Therefore, the search for the prerequisite structure between KCs in a student model is a critical aspect of ITS. Several techniques have been proposed to address this problem. We focus on the techniques for searching the prerequisite structure between knowledge components in a domain model.

2.3.1 Defining an ordering between test items or KCs

Some models have introduced the concept of learning paths, which are suggested orderings between test items. This has defined a mathematical framework for ordering learning objects and, by extension, between knowledge components.

Knowledge Space Theory

The Knowledge Space Theory (KST), proposed by Doignon and Falmagne, relies on test item structures [DF12]. It supposes the learner's knowledge is defined as a set of knowledge items, and these test items can only be mastered in a constrained order. In other words, KST defines an ordered structure among test items, called a lattice, which is a sort of prerequisite relationship between items. Nevertheless, KST does not provide a data-driven procedure for deriving the ordering between items or skills. It only relies on algorithms that extract the ordered structure of items from expert domain knowledge, such as the one introduced by Koppen et al. [KD90].

ALEKS is a math and science learning environment that uses KST to adapt to a student's knowledge [Can01] and provides individualized assessments [FRHG19]. ALEKS uses adaptive questioning and machine learning algorithms to assess a learner's knowledge in various subjects, including math, science, and business.

Competence-based Knowledge Space Theory

Similarly, an extension to this theory, namely Competence-based Knowledge Space Theory (CbKST), discusses how to integrate the domain model to KST by incorporating the cognitive level of the "competencies" and their precedence links into the theory [HSHA06, HAH+13]. A competence in CbKST is similar to a knowledge component and a precedence link to a prerequisite relationship. Thus, the competence structure or the lattice generated from the

domain model allows for relating the learner knowledge states instead of items in KST. This greatly improves the potential of systems to adapt.

2.3.2 Search of prerequisite KC from statistical tests

Knowledge Space Theory and its extensions provide a framework for identifying statistical implications expected to arise in data. Statistical tests have been devised to verify the presence of these implications and establish the underlying relationships between test items or knowledge components.

Bayesian networks

The notion of the prerequisite relationship between knowledge components is deeply related to causality, as depicted in Definition 4.1.1. Consequently, the trivial paradigm for studying such relationships is the Bayesian framework, which can be used to evaluate the probability of prerequisite structures, and that will be studied further in Chapter 3. A priori knowledge of the domain prerequisite structure has been integrated into simple learner models, most of the time with Bayesian Networks (BNs) [CGV02, CMPdlC+05]. These techniques typically involve experts using their domain knowledge to define the prerequisite relationships between the KCs. Recent works employ data to retrieve the conditional probabilities that rule such BNs [DMG06]. Still, they do not consider the evolution over time of learners' knowledge states.

Partial-Order Knowledge Structures

Desmarais et al. study various uses of the Bayesian framework to learn test item structures and predict the learners' performance from evidence [DMG06]. Partial-Order Knowledge Spaces (POKS) rely on the naive Bayes framework with additional constraints induced by the closure under union and intersection assumed in KST. POKS show better results than usual Bayesian network approaches and are less intensive computationally speaking, thanks to the constraints. The approach entailed by POKS is the same as classic Bayesian networks because it studies the causality expressed in the data to refine the structure. Thus, like classic Bayesian networks, POKS do not consider the evolution over time of the learner's knowledge state and miss the chance to consider the learning and forgetting phenomena and their incidence on the study of the prerequisite structure. Pavlik et al. have applied additional statistical tests on covariance matrices to recreate the prerequisite KC structure from the POKS structure learning procedure [PCWK08]. They do not consider the dynamicity of the learning process either.

Statistical tests on latent variables computed from performance prediction

From now, the study of correlation or, sometimes, causality between elements of knowledge has been realized from observable variables. We present works that first preprocessed learners'

transactional data to define latent variables more related to knowledge components.

Chen et al. have introduced Probabilistic Association Rules Mining, a technique to discover association rules from uncertain data on preprocessed learner transaction data [CWL15]. The preprocessing aims to discover learners' knowledge states so that they can apply statistical techniques to latent variables.

Similarly, Scheines et al. considered latent variables instead of observable ones to conduct statistical tests on the prerequisite structure [SSG14]. They assume a known Q-matrix, which is a tool for mapping the test items with the knowledge components. They search for the prerequisite structure by performing a causal discovery algorithm on the correlation matrix between latent variables, determined with a measurement model.

LiFT, introduced by Pavlik et al., proposes representing the KC structure through the Q-matrix in PFA [JCK09]. They apply statistical tests on the performance of PFA with any possible prerequisite structure and recreate the whole structure by studying test items pairwise. Finally, Piech et al. use Deep Knowledge Tracing outcomes to perform statistical tests on the prerequisite KC structure [PBH+15].

These statistical tests have the advantage of being easy to compute, as numerous algorithms exist for predicting the learners' performance and proficiency. Nevertheless, we have reported in Section 2.2.4 the lack of interpretability of these underlying techniques, such as logistic regression or deep learning.

2.3.3 Searching the KC structure from parameter fitting procedures

The techniques presented so far study the prerequisite KC structure from statistical tests performed on direct outcomes. These techniques are applied to data on observable variables, or to preprocessed data representing the predicted learners' knowledge states. However, some learners' performance prediction techniques directly include the existence of the prerequisite structure of the domain model. Parameter fitting techniques can be directly employed to discover the prerequisite KC structure.

Q-matrix fitting

Pardos et al. introduce direct Q-matrix refinement in the model parameter fitting procedure [PD+18]. They use an augmented variant of the Additive Factors Model (AFM) algorithm, which integrates prerequisite constraints when predicting the learner's performance. The learned Q-matrix can be seen as a representation of the prerequisite KC structure, as did Pavlik et al. However, it remains a study of the test item structure rather than a KC structure.

Knowledge tracing

Finally, model-tracing learner models and, more specifically, knowledge-tracing models may include the effect of prerequisite relationships when inferring the learner's knowledge state. It is notably the case in Käser et al.'s work. They integrate the causal effect of prerequisite relationships between KCs in their Dynamic Bayesian Network (DBN) [KKSG17]. Still, because of the lack of interpretability reported in Section 2.2.4, the parameters obtained from training on data are not necessarily in line with the parameters of the Bayesian network. While this approach has an explicit limitation, namely the lack of interpretability, it considers the influence of learning and forgetting phenomena which are crucial when studying the structure between KC masteries. Developing an interpretable procedure for discovering the prerequisite KC structure that integrates learning and forgetting phenomena is a challenging task that will be the focus of our work.

2.4 Discussion

In conclusion, this chapter has aimed to provide a comprehensive overview of the ITS components. In particular, we have provided a deep focus on learner models, which can be of multiple types. Additionally, we have explored the different perspectives to model students and presented rule-based and data-driven approaches to learner modeling. We have highlighted hybrid learner models that combine both.

Furthermore, these models can be used to predict the student's performance. We have discussed the two main families of learners' performance prediction algorithms: logistic regression algorithms and cognitive diagnosis algorithms. We have highlighted the importance of interpretability in these algorithms, and how the top-performing algorithm in terms of performance prediction can be the least interpretable model. This has motivated our choice to continue with Bayesian networks, which allow for better parameter interpretability than any other machine learning technique.

Finally, we have explored the various approaches for searching the prerequisite KC structure of the domain model. In particular, we have seen that most of these techniques rely on statistical tests and do not assert the causality implied by prerequisite relationships.

Chapter 3

Bayesian networks, a model for knowledge-enhanced machine learning

In Chapter 2, we have demonstrated how learner modeling is a subject of interest, especially in ITS. Capturing the complex interplay between the multiple variables influencing the learner's knowledge is key. The learner's knowledge can be seen as a complex system, with many interacting and interdependent components that work together. Overlaying the learner model on top of the domain model provides an additional layer of complexity to modeling learners' knowledge. The learner is modeled regarding their cognitive processes and in relation to the domain model representing the knowledge being learned. This approach assumes that the domain model provides a structured representation of the knowledge being learned. The learner's understanding of the domain constantly evolves as they interact with the learning environment.

We focused on symbolic and data-driven approaches for modeling the learner's knowledge and highlighted their differences. On the one hand, techniques from Knowledge Representation and Reasoning (KRR), such as ontologies or rule-based systems, can be applied to reason about the learner regarding the domain model. These techniques can be used to capture the structure of the domain and to represent the relationships between different concepts and ideas within the domain. On the other hand, data-driven machine-learning techniques, such as logistic regression or neural networks, can also be used to model the learner without explicitly reasoning on the domain knowledge. These techniques rely on large datasets of examples to identify patterns and relationships that can be used to make predictions about the learner's transactions with the system. While these models may be less interpretable than KRR-based models, they can be more flexible and adapt more readily to changes in the learning environment. Nevertheless, we figured that knowledge representation techniques could be integrated into machine learning models, combining the strengths of both approaches by incorporating the domain knowledge as prior probabilities or constraints into the learner model.

In this chapter, we focus on Bayesian Networks (BNs). The Bayesian network framework enables the mix of symbolic AI and machine learning paradigms to benefit from the strengths of both approaches. BNs will be the central tools employed in our thesis work. Therefore,

we aim to provide a transversal comprehension of this technology. We start by studying the techniques for representing uncertainty in domain representation with BNs. More particularly, we present a quick overview of the probability theory, which defines metrics for measuring uncertainty. We introduce BNs as Probabilistic Graphical Models (PGMs) that provide a mathematical formulation of the dependencies in multi-agent systems, widely used in fields such as natural language processing [Rat96], computer vision [Li09], and bioinformatics [BB01]. Then, we present how to reason on BN knowledge representation and how data can be included in the reasoning process. We introduce the concept of inference in BNs, and we present state-of-the-art techniques for exact and approximate inference. Finally, we provide an overview of the BN parameter learning techniques and specify the context in which they can be applied.

3.1 Including uncertainty in knowledge representation with Bayesian networks

Bayesian Networks (BNs) provide a powerful framework for including uncertainty in Knowledge Representation and Reasoning (KRR). Uncertainty arises in many real-world applications, especially when the available data is incomplete or subject to noise. Probability theory can be used to represent and reason about uncertain knowledge. Probabilistic Graphical Models (PGMs) are a natural way to represent probability distributions over complex systems. They provide a compact and intuitive way to represent the joint probability distribution over a set of random variables. BNs, a kind of PGM that explicitly represent the conditional dependencies between variables, are proper tools for mixing symbolic and data-driven approaches for modeling the learner's knowledge.

In this section, we will introduce the basics of Bayesian networks, namely uncertainty representation with probability theory and probabilistic graphical models. We will also discuss the use of data in PGMs to introduce BNs as knowledge-enhanced machine-learning tools.

3.1.1 Representing the knowledge uncertainty with probability theory

Domain knowledge can be decomposed into explicit knowledge, implicit knowledge, and beliefs [KF09]. This indicates that describing the knowledge of a system must take uncertainty into account. Uncertainty may arise from several elements: attribute uncertainty, class uncertainty, and structural uncertainty. Attribute and class uncertainty relates to the uncertainty on the state of the system concepts. It could speak for the ignorance of the system components' states or the lack of precision in the measurements. Structural uncertainty corresponds to the uncertainty in the relationships. It may represent the effectiveness of a relationship between concepts and potentially translate the expert beliefs on this relationship. Taking uncertainty into account allows for representing incomplete or uncertain information, which is often the case in real-world scenarios, especially concerning beliefs experts may express on the system. This enables more accurate and realistic modeling of the underlying knowledge and can lead to more robust and reliable decision-making.

Probabilities to express uncertainty

Probability theory is a tailored tool to indicate the uncertainty of the system knowledge. It provides a mathematical framework for modeling uncertainty. It enables a precise and quantitative representation of the degree of uncertainty, assigning probabilities to the different states of the knowledge representation components. It offers a way to reason under uncertainty.

Probability theory is a mathematical framework for modeling uncertain events. We introduce some basics of probability theory to define the concept of probability distributions. Probability

theory relies on measuring sample space and event space to define probability. A sample space is a set of all possible outcomes of an experiment or a random process. For example, if we flip a coin, the sample space would be $\{heads, tails\}$.

Definition 3.1.1 (Sample space)**.** *The sample space Ω is the set of all experiment outcomes. Every element $\omega \in \Omega$ fully describes the state of the complex system at the end of the experiment.*

An event is a subset of the sample space. It represents a specific outcome or combination of outcomes. For example, the event "*heads*" would be the subset $\{heads\}$ of the sample space $\{heads, tails\}$. The event space is the set of all possible events that can occur. In knowledge representation, events can represent the states of different components of the knowledge representation. Attributes, classes, and structures are depicted with events that describe them.

Definition 3.1.2 (Event space)**.** *The event space F associated with the sample space Ω is the set whose elements $A \in F$, also known as events, are subsets of Ω, i.e. $A \subseteq \Omega$.*

The probability of an event is a measure of how likely the event is to occur. It is defined as the ratio of the number of favorable outcomes to the total number of possible outcomes and relates to uncertainty in the state of events. We define the probability measure in Definition 3.1.3.

Definition 3.1.3 (Probability measure)**.** *The probability measure of an event space F associated to the sample space Ω is the function $P : F \to \mathbb{R}$ such that:*

- $\forall A \in F,\ P(A) \geq 0$
- *Let be $A_1, ..., A_n$ such that $\forall\ i \neq j,\ A_i \cap A_j = \varnothing$, then $P(\cup_{i \in [\![1,n]\!]}\ A_i) = \sum_{i \in [\![1,n]\!]} P(A_i)$*
- $P(\Omega) = 1$

These three properties are called the probability axioms, also known as Kolmogorov axioms [Kol13]. They entirely define the probability measure. Other properties are verified by the probability measure and will be defined afterward. Still, we can note that supposing a studied system can be fully described with probabilities on the event space is a strong hypothesis. It is sometimes not even possible to assign probabilities to events.

> *"There are known knowns;*
> *These are things we know that we know.*
> *There are known unknowns;*
> *That is to say,*
> *There are things that we now know we don't know.*
> *But there are also unknown unknowns;*
> *There are things we do not know we don't know."*
> — *Donald Rumsfeld*

Random variables

Supposing the knowledge is represented by concepts and relationships, we might want to define functions that give the probability measure from the state of these elements. We introduce the concept of random variables in Definition 3.1.4 to track the state of these concepts and to quantify the uncertainty to which they are subject.

Definition 3.1.4 (Random variable)**.** *A random variable X is a function $X : \Omega \to E$, where E is some measurable space. The value $X(\omega)$ that the random variable X takes for a random outcome $\omega \in \Omega$ is denoted x.*

A random variable fully describes the probabilities associated with a set of events. For an element of knowledge representation, one might consider the sample space as all the possible combinations of states of its attributes. Random variables are the theoretical support for defining the probability measure on the concepts describing the system knowledge.

Random variables may be of multiple types. In this work, we study discrete random variables, where the measurable space E takes its values in $\mathbb{N}$. More particularly, we consider binary random variables, which are random variables where the measurable space is $\{0, 1\}^n$, where n is the dimension of the sample space Ω. Both attribute and class uncertainty on knowledge is expressed with random variables.

Probability distribution

Given a random variable X describing a concept of the system knowledge, the outcomes taken by X are the elements of the sample space, and the uncertainty of concepts is quantified through the probability of each outcome of the sample space. Because the random variables are supposed to be discrete, the probabilities of all possible outcomes of X compose the probability distribution of X. It is directly related to the probability mass function p_X. The probability mass function p_X is the probability distribution of a discrete random variable X.

Definition 3.1.5 (Probability mass function)**.** *Let be X a discrete random variable. The probability mass function of X is the function $p_X : \mathbb{R} \to [0, 1]$ such that*

$$p_X(x) = P(X = x)$$

On the other hand, knowledge structure uncertainty corresponds to uncertainty in the relationships between concepts. These relationships between concepts express the dependence between the random variables that represent the concepts. Events, such as random variable outcomes, can be conditionally related. We introduce the notion of conditional probability in Definition 3.1.6.

Definition 3.1.6 (Conditional probability)**.** *Let B be an event such that $P(B) \neq 0$. The*

conditional probability of any event A given B is defined as

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{3.1}$$

Conditional probabilities are useful tools when studying the problem of knowledge structural uncertainty. Indeed, the uncertainty in the relationship between two knowledge concepts can be represented with a Conditional Probability Distribution (CPD) that regroups the conditional probability between every possible outcome of the two random variables (see Section 3.1.2 for more details on CPDs). When an event does not change the probability of another, the two events are said to be independent.

Definition 3.1.7 (Independent events)**.** *Let be A and B two events. A and B are independent events if $P(A \mid B) = P(A)$ or if $P(B) = 0$.*

Notation 3.1.1 (Independent events)**.** *A and B are independent events is denoted $A \perp\!\!\!\perp B$.*

Consequently, according to the definition of conditional probability in Definition 3.1.6, the probability of two independent events equals the product of the probability of each event. The equivalence is reported in Property 3.1.1.

Property 3.1.1 (Independence and conditional probability)**.**

$$\forall A, B \ s.t. \ A \perp\!\!\!\perp B \iff P(A \cap B) = P(A)P(B) \tag{3.2}$$

3.1.2 Probabilistic graphical models

Köller et al. present one of the main advantages of using probability theory in modeling complex systems, compared to first-order logic approaches [KF09]. The range of outcomes can be too wide to be fully considered. Probabilities allow for avoiding the annoying exceptions and exceptional cases when their probability exhibits that they are unlikely to happen. Probabilistic Graphical Models (PGMs) are KRR approaches that use probability distributions to describe system knowledge.

A handful and exploitable model

The main idea of PGMs is to reduce the dimension of the probability distribution of the whole system knowledge by designing a simplified model with probabilities. It benefits from the dependence structure between the system elements to factorize the distribution into several small local distributions. Then, it mobilizes mathematical, statistical, and computational tools to compute them.

Factorization of the whole distribution is allowed by studying the dependencies between the variables that describe the system. It is essential to figure this relation out. The set of dependencies that rule the system directly impacts the distribution structure. Conversely,

when factorizing is observed in this distribution, the system follows a set of independence. PGMs represent as a graph the set of dependencies between the elements of the system knowledge. This representation is helpful in regards to the understanding of the model, thanks to the great readability of graphs.

Definition 3.1.8 (Probabilistic graphical model)**.** *A Probabilistic Graphical Model is a type of mathematical model for representing and reasoning about uncertain knowledge. It uses a graph structure to represent the relationships between random variables representing uncertain or unknown quantities. Each node in the graph represents a random variable, and edges between nodes represent dependencies or relationships of independence between the variables.*

PGMs are used to represent probability distributions over the variables of interest, and they provide a powerful tool for reasoning about uncertain information. Their graph structure allows for a compact and intuitive representation of complex probability distributions.

Principles of probabilistic graphical models

PGMs exploit the structure to describe complex probability distributions in a compact way. They fully depict the system by defining the interactions between all its elements by local probabilistic dependence. They introduce conditional independence to summarize the dependencies between the random variables.

Definition 3.1.9 (Conditional independence)**.** *Let be A, B and C three events. A and B are conditionally independent events given C, if $P(A \mid B \cap C) = P(A \mid C)$ or if $P(B \mid C) = 0$.*

Notation 3.1.2 (Conditional independence)**.** *The conditional independence of A and B given C is denoted $A \perp\!\!\!\perp B \mid C$.*

Such as the independence property on conditional probability, we can rewrite the definition of conditional independence as in Property 3.1.2.

Property 3.1.2 (Conditional independence and conditional probabilities)**.**

$$\forall A, B, C \text{ s.t. } (A \perp\!\!\!\perp B \mid C) \iff P(A \cap B \mid C) = P(A \mid C)P(B \mid C) \tag{3.3}$$

The set of conditional independencies between the elements of a complex system is represented by a graph in probabilistic graphical modeling. The graph summarizes all the local interactions in the system. It is a graphical representation of the set of local Conditional Probability Distributions (CPDs) translating the dependencies between the variables.

Classic conditional probability distributions

CPDs aim at describing the dependencies between the variables of a model. For each random variable X of the PGM, we define $P_{\mathbf{pa}_X}$ as the conditional probability distribution of X

given the state $\mathbf{pa}_X$ of variables $\mathbf{Pa}_X$. We have the following properties for $P_{\mathbf{pa}_X}$.

Definition 3.1.10 (Conditional probability distribution)**.** *Let be X a random variable and $\mathbf{pa}_X$ the set of random variables such that X is dependent on $\mathbf{Pa}_X$. The conditional probability distribution of X given $\mathbf{pa}_X$ is the function $P_{\mathbf{pa}_X} : \mathbb{R} \to [0, 1]$ such that*

$$P_{\mathbf{pa}_X}(x) = P(X = x \mid \mathbf{pa}_X) \tag{3.4}$$

It verifies

$$\begin{cases} \forall x \in Val_X, P_{\mathbf{pa}_X}(x) \geq 0 \\ \sum_{x \in Val_X} P_{\mathbf{pa}_X}(x) = 1 \end{cases} \tag{3.5}$$

CPDs result in the factorization of the joint distribution implied by the set of conditional independencies. Various kinds of CPDs can model these relationships between variables. The choice among them will affect the knowledge reasoning approach. The most trivial representation of $P_{\mathbf{pa}_X}$ is the deterministic CPD, defined in Definition 3.1.11.

Definition 3.1.11 (Deterministic CPD)**.** *A CPD is deterministic if there exists a deterministic function $f : Val_{\mathbf{Pa}_X} \to Val_X$ such that*

$$P(x \mid \mathbf{Pa}_X = \mathbf{pa_X}) = \begin{cases} 1 & \text{if } x = f(pa_X) \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

Because we study binary variables in this work, only logical functions will be detailed here. We resume the main logical functions that can be employed as the deterministic function f in Table 3.1. Nevertheless, we must keep in mind that other deterministic functions can be applied to other kinds of variables. For instance, algebraic functions are usually employed for ordinal variables.

	Deterministic function f
NOT	$f(x) = \neg x$
OR	$f(\mathbf{x}) = \cup_{x \in \mathbf{X}} x$
AND	$f(\mathbf{x}) = \cap_{x \in \mathbf{X}} x$

Table 3.1: Main deterministic functions used in deterministic CPDs. The NOT function only applies to a unique variable x, while OR and AND functions can be applied to multiple variables $\mathbf{x}$.

We note that deterministic CPDs do not require numerical parameters, as the outcomes are fully computable from f. They perfectly describe some phenomena in nature, such as the

phenotype of a person from her genotype. Modeling the behavior of electronic circuits also particularly fits with deterministic CPDs: machine-oriented models are easy to implement with deterministic CPDs, as the components of these systems use deterministic functions.

However, despite these few examples, Díez et al. point out that deterministic relationships are not very common in practice because of the uncertainty of real-world interactions [DD06]. Still, using deterministic variables in a PGM can reduce the complexity of a model. Deterministic relationships may occur naturally in modeling many domains, but they are mainly used to simplify the dependencies in a complex model [KF09]. In fact, a deterministic CPD in a PGM implies a case of Context-Specific Independence (CSI), as the dependencies between the variables describing the PGM depend on the context.

Definition 3.1.12 (Context-specific independence)**.** *Let* $\mathbf{X}$*,* $\mathbf{Y}$*,* $\mathbf{Z}$*,* $\mathbf{C}$ *be pairwise disjoint sets of variables.* $\mathbf{X}$ *and* $\mathbf{Y}$ *are contextually independent given* $\mathbf{Z}$ *and the context* $\mathbf{c} \in Val(\mathbf{C})$ *if*

$$P(\mathbf{X} \mid \mathbf{Z}, \mathbf{c}, \mathbf{Y}) = P(\mathbf{X} \mid \mathbf{Z}, \mathbf{c}) \text{ whenever } P(\mathbf{Y}, \mathbf{Z}, \mathbf{c}) > 0 \tag{3.7}$$

Deterministic CPDs are examples of simple CSI, but other kinds of CPDs encode more complex CSI. For instance, rule-based CPDs rely on explaining how the system works from a set of hypothetical rules. Rule-based models aim to reproduce the behavior observed in the real world with scenarios and rules.

Definition 3.1.13 (Rule-based CPD)**.** *A rule-based CPD* $P(X \mid \mathbf{Pa}_X)$ *is a set of rules* $\mathcal{R}$ *such that:*

- *Each rule* $\rho \in \mathcal{R}$*, which is a pair* $\langle \mathbf{c}, p \rangle$ *with* $\mathbf{c}$ *an assignment to some subset of variables* $\mathbf{C} = Scope[\rho]$ *and* $p \in [0, 1]$*, verifies* $Scope[\rho] \subseteq \{X\} \cup \mathbf{Pa}_X$
- *For each assignment* $(x, \mathbf{u})$ *to* $\{X\} \cup \mathbf{Pa}_X$*, there is precisely one rule* $\langle \mathbf{c}, p \rangle \in \mathcal{R}$ *such that* $\mathbf{c}$ *is compatible with* $(x, \mathbf{u})$*. In this case, we have* $P(X = x \mid \mathbf{Pa}_X = \mathbf{u}) = p$*.*

The rule-based CPD $P(X \mid \mathbf{Pa}_X)$ *verifies* $\sum_{\mathbf{u}} P(x \mid \mathbf{u}) = 1$*.*

In practice, this approach would imply writing computer programs for context purposes. That is, designing specific models for every complex system one wants to model and every question one wants to answer. Intuitively, as fruitful as the result of such a model, it remains a result with a narrow scope of applicability. One of the goals of modeling is to provide enough generic solutions. It would require significant efforts to apply rule-based models to other systems or research questions than the ones on which they were designed. Tabular CPDs, also known as Conditional Probability Tables (CPTs), are tables that contain all the information about the CPDs. More precisely, it encodes $P(X \mid \mathbf{Pa}_X)$ as a table. We define tabular CPDs in Definition 3.1.14.

Definition 3.1.14 (Tabular CPD)**.** *A tabular CPD is the encoding of the conditional probability distribution* $P(X \mid \mathbf{Pa}_X)$ *as a table that contains a nonnegative entry for each*

joint assignment to X *and* $\mathbf{Pa}_X$*. For each assignment* $\mathbf{pa}_X$ *of* $\mathbf{Pa}_X$*, the tabular CPD verifies*

$$\sum_{x \in Val(X)} P(x \mid \mathbf{Pa}_X = \mathbf{pa}_X) = 1 \tag{3.8}$$

ICI models

As mentioned earlier, deterministic CPDs are unrealistic but can be included in complex distributions to reduce the number of parameters. Some models are based on the clause of Independence of Causal Influences (ICI). ICI models suppose that there are no interactions among the causal mechanisms by which the parents of a node affect its value [DD06].

Definition 3.1.15 (ICI-model CPD)**.** *Let* X *be a random variable with* k *parents denoted* $Pa_{X,1}$, ..., $Pa_{X,k}$.

The CPD $P(X \mid Pa_{X,1}, ..., Pa_{X,k})$ *exhibits independence of causal influence if we can introduce auxiliary variables* Z_1, ..., Z_k *respectively attached to* $Pa_{X,1}$, ..., $Pa_{X,k}$ *such that each* $Pa_{X,i}$ *is conditionally independent with* X *given* Z_i *and with other* $Pa_{X,j}$ *given all* $(Z_k)_k$ *for* $j \neq i$*.* $\forall i$*, the CPD of* Z_i*s are deterministic functions* f_i *of* X_i*s, and the CPD of* Y *is also a deterministic function of the* Z_i*s. It can be represented as in Figure 3.1.*

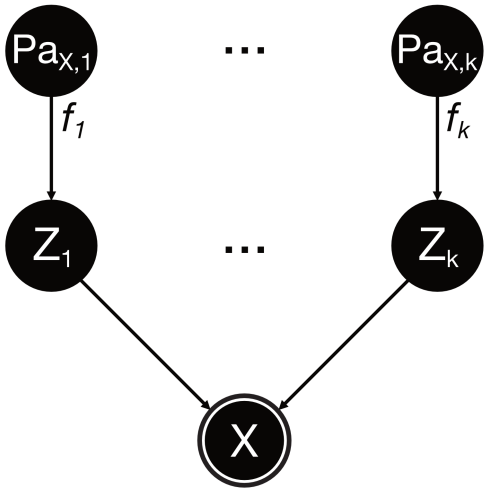


Figure 3.1: Representation of the ICI-model CPD of X and its parents $Pa_{X,1}, ..., Pa_{X,k}$.

The deterministic function defining X is generally an AND or an OR function [DD06]. The corresponding ICI-model CPDs are respectively called Noisy-AND and Noisy-OR CPDs.

Number of parameters

Every CPD representation discussed above can be represented with a CPT. However, the tabular representation requires a significant number of parameters, as the number of parameters needed to describe a CPT depends on the number of possible outcomes of

$\mathbf{Pa}_X$. For binary variables, there are $2^{|Val(\mathbf{Pa}_X)|}$ parameters for the CPT of X. The CPT representation can be difficult to handle when there is a large number of parents.

Example 3.1.1 (Number of parameters in the different CPD representations)**.** *Let's suppose we study the Grade (G) of a student for an exam. It can take values from A to F. We suppose two other variables are studied in this example: Study Habits (H) and Absenteeism (A), and both are binary.*

Using a deterministic CPD to define G from H, we can represent the relationship with a simple table with two columns (study habits and exam performance), and two rows (good and poor). This CPD would require only 2 parameters.

H	G
Good	A
Poor	F

Table 3.2: Example of a deterministic CPD for G.

A rule-based CPD representation of Table 3.2 would, for instance, include a condition on A. Then, with a rule-based CPD, the number of parameters increases to 4, as we need to specify the rules for good and poor study habits for each possible level of absenteeism:

H	A	G
Good	Low	A
Good	High	B
Poor	Low	C
Poor	High	F

Table 3.3: Example of a rule-based CPD for G.

With a tabular CPD, the number of parameters increases dramatically. We need to specify the probability of each outcome for every possible combination of study habits, absenteeism, and exam performance. In this example, the tabular CPD would require 8 parameters.

The number of parameters required for a tabular CPD increases exponentially as the number of parents increases. This can make it difficult to handle when there is a large number of parents. In the early days, the parameters were indeed specified by experts in the modeled system [BL04]. Defining an enormous number of parameters, which may be hard to retrieve, is nearly an impossible task to ask.

Nevertheless, we note that the number of parameters required for a CPD does not always

H	A	G	Probability
Good	Low	A	0.8
Good	Low	B	0.2
Good	High	A	0.6
Good	High	B	0.4
Poor	Low	C	0.7
Poor	Low	F	0.3
Poor	High	C	0.5
Poor	High	F	0.5

Table 3.4: Example of a tabular CPD for G.

directly correspond to the complexity of the model and that other factors, such as the structure of the model, also play a role. Moreover, there are other downsides to using CPTs (e.g., the discretization problem) that will not be considered in this work, as we only consider binary variables.

The ICI-model CPD benefits from parameter reduction of deterministic relationships. The number of its parameters grows linearly with the number of node parents. While the number of parameters of a tabular CPD of X for a node and its n parents would be 2^n, the number of parameters of an ICI-model CPD is $2n$.

3.1.3 Bayesian networks, a kind of PGM

Bayesian Networks (BNs) are a kind of Probabilistic Graphical Model (PGM). They suppose that the distribution of the whole can be decomposed thanks to the conditional dependencies of the system.

A graphical representation of the conditional independencies of the distribution

A Bayesian network is represented by a Direct Acyclic Graph (DAG).

Definition 3.1.16 (Direct acyclic graph)**.** *A direct acyclic graph $\mathcal{G}(V, E)$ is a graph composed of a set of vertices V connected among them by directed edges E without directed cycles.*

A Bayesian network is then a directed graphical model. The structure of the DAG that represents the Bayesian network induces the factorization of the whole distribution. It is allowed by the set of assumptions about the dependencies between the variables that describe the system. The structure of the DAG depends on the conditional independencies between

the variables. Each of its edges indeed represents a conditional dependence of two variables. Conversely, the choice of the DAG will impact the assumptions on the distribution.

In particular, if two variables are d-separated relative to a set of variables Z in a directed graph, they are independent conditional on Z in all probability distributions such a graph can represent. d-separation is a criterion for determining whether two variables in a directed graphical model are independent, given the presence of certain other variables.

Definition 3.1.17 (d-separation)**.** *Given a DAG $\mathcal{G}(V, E)$ and a set of variables $\boldsymbol{Z}$ subset of V, two variables X and Y in V are d-separated given $\boldsymbol{Z}$ if and only if there is no active path between X and Y in $\mathcal{G}$ given $\boldsymbol{Z}$.*

An active path is a path that starts at X, ends at Y, and contains no collider (a node with two incoming edges) that is not blocked by a variable in $\boldsymbol{Z}$. If X and Y are not d-separated, they are d-connected.

The graph structure of a DAG shows simple characteristics of d-separation.

Property 3.1.3 (d-separation in a DAG)**.** *The minimal set of nodes that d-separates a node $v \in V$ of the DAG $\mathcal{G}(V, E)$ is the Markov blanket of the node in the DAG.*

Definition 3.1.18 (Markov blanket in a DAG)**.** *The Markov blanket of a node in a DAG consists of the set of nodes containing its parents, its children, and any other parents of its children in the DAG.*

To put it in a nutshell, Bayesian networks are the combination of a DAG structure with probability theory.

Definition 3.1.19 (Bayesian network)**.** *A Bayesian network is a directed graph $\mathcal{G}(V, E)$ associated to a set of $n = |V|$ random variables $(X_i)_{i \in [\![1,n]\!]}$ and CPDs $(P_{\mathbf{pa}_{X_i}})_{[\![1,n]\!]}$. Its probability mass function is*

$$p(\mathbf{X} = \mathbf{x}) = \prod_{i \in [\![1,n]\!]} P\left(X_i = x_i \,\middle|\, \mathbf{x}_{\mathbf{pa}_{X_i}}\right) \tag{3.9}$$

BNs consider one random variable and one CPD per node. They exploit the chain rule of conditional probabilities to compute the whole distribution.

Property 3.1.4 (Chain rule)**.** *Let $S_1, ..., S_k$ be events. The chain rule states that:*

$$P(S_1 \cap ... \cap S_k) = P(S_1) \times P(S_2 \mid S_1) \times ... \times P(S_k \mid S_1 \cap ... \cap S_{k-1}) \tag{3.10}$$

In particular, we note that, for $k = 2$ events, the chain rule corresponds to the definition of conditional probability. In practice, the chain rule only consists in applying the conditional probability definition several times in a row.

The strong assumptions on conditional independence between variables imply that the distribution of the whole is fully known from them. Given the chain rule, the probability mass function can be formulated as the product of conditional probability distributions. The distribution of the whole can be fully described by local CPDs. One of the key advantages of BNs is their capacity to decompose the joint distribution of a system into a set of tractable conditional distributions. In large BNs, the sample space can be extensive, making the computation of the joint distribution intractable. However, by using the structure of the BN to factorize the distribution, the computational complexity can be significantly reduced.

Furthermore, the graphical representation of BNs provides a level of transparency that allows human experts to easily understand and evaluate the structure and the CPDs of the model. This transparency is crucial for constructing models that effectively capture the underlying system and provide a clear understanding of the modeled domain. Without this transparency, the results of the model may be difficult to interpret or may not align with users' expectations.

Simple examples of Bayesian networks

Let us introduce a toy example to understand how it works. The SAT Reasoning Test is a standardized national exam in the USA used for university admission. One can introduce a simple model to compute the student's intelligence from her score on SAT. Let be two binary random variables Intelligence (I) and SAT score (S). We suppose that the SAT score depends on the student's Intelligence and that these two variables are binary. We have the Bayesian network represented in Figure 3.2.

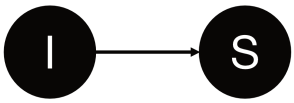


Figure 3.2: Bayesian network that model the SAT score (S) as a function of the student's intelligence (I).

The joint distribution of the random variables I and S is composed of four entries, that are $\big((I = 0, S = \mathit{fail}), (I = 0, S = \mathit{pass}), (I = 1, S = \mathit{fail}), (I = 1, S = \mathit{pass})\big)$. Using the Bayesian network grants the reduction of the required information to compute the joint distribution. Instead of considering $P(I, S)$, which is the distribution of the whole, the Bayesian network in Figure 3.2 only requires prior knowledge of I and conditional probability distribution of S given I. Root nodes in BNs are the nodes with no parents, that is to say, no incoming arcs. They are informed by prior knowledge, while conditional probability distributions inform other nodes. The prior knowledge in our example is $P(I)$. The CPD is $P(S|I)$.

BNs are, in practice, much wider than the toy example presented in Figure 3.2. Besides, we introduce three additional random variables to represent a more complex BN: the course Difficulty (D), Grade (G), and the quality of the recommendation Letter (L). D and L

are also binary random variables, while G is a three-valued discrete random variable. The Grade depends on both the course Difficulty and the student's Intelligence. The quality of the recommendation Letter is supposed to be uniquely dependent on the student's Grade. This example is widely used to present Bayesian networks and is called the *Student* example. We represent the DAG of such BN in Figure 3.3. Its structure gives information on the conditional dependencies between the variables that compose the BN.

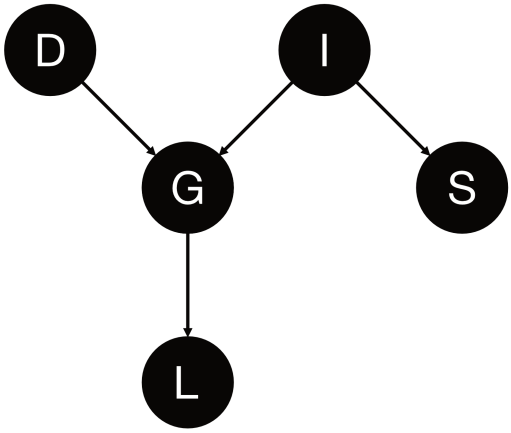


Figure 3.3: DAG structure of the Bayesian network representing the *Student* example.

The Bayesian network is fully specified if the CPD of each BN node is given. We saw in Section 3.1.2 that tabular CPDs are particularly relevant for Bayesian networks. Table 3.5 shows CPDs that could be associated with the DAG of Figure 3.3.

		G		
I	D	low	medium	high
0	0	0.1	0.4	0.5
0	1	0.05	0.3	0.65
1	0	0.2	0.5	0.3
1	1	0.1	0.4	0.5

(a) CPT of G

	S	
I	fail	pass
0	0.2	0.8
1	0.1	0.9

(b) CPT of S

	L	
G	bad	good
low	0.8	0.2
medium	0.6	0.4
high	0.2	0.8

(c) CPT of L

Table 3.5: Examples of CPD that could be attached to the DAG represented in Figure 3.3.

The combination of the DAG in Figure 3.3 and the set of CPTs given in Table 3.5 are sufficient to fully define the joint distribution of the random variables D, I, G, S, and L, and to compute any marginal probability distribution that can be extracted from it. We will detail the task of inference in Section 3.2.

Computational implementation of Bayesian networks

From a computational perspective, Bayesian networks are the source of many libraries in almost every programming language. We use `Python` in this work. There are plenty of packages available in `Python` for inference with Bayesian networks.

- `pyAgrum` is a `Python` package of Bayesian networks [DGW20]. It derives from the `C` package `aGrUm`. It includes plenty of features, such as dynamic Bayesian networks. It is the package we use in this work.
- `pgmpy` is a `Python` package of PGMs, that includes Bayesian networks.
- `pomegranate` is also a `Python` package of PGMs, including Bayesian networks.

There are, of course, other open-source implementations of Bayesian networks in `Python`.

3.1.4 Dynamic Bayesian networks

Bayesian networks consider a fixed set of variables $\mathbf{X}$. However, most complex systems cannot be modeled with variables at a fixed time, as the state of the concepts that compose the complex system evolves over time. The complex system is then dynamic. We assume time can be discretized into timeslices t. In order to represent the distributions of systems in which the state varies over time, we introduce Dynamic Bayesian Networks (DBNs).

A compact representation for large repetitive Bayesian networks

A dynamic Bayesian network $\mathcal{B}$ is a dynamic graphical model that is composed of time-dependent variables. Instead of considering time-independent random variables $\mathbf{X}$, the DBN variables are indexed with time. $\forall t \in [\![0, T]\!]$, $\mathbf{X}^{(t)} = \left\{X_i^{(t)}\right\}_i$ represent the same events at different timeslices. The idea is to provide a Bayesian network that replicates the variables through time. A DBN can then be seen as the concatenation of multiple duplications of the same Bayesian network side by side that would be linked with each other. Its variables are $\{X_i^{(t)} \mid i \in [\![0, N]\!], t \in [\![0, T]\!]\}$, with N the number of nodes for a timeslice and T the temporal length of the DBN. The temporal length is a parameter and will not change the structure of the DBN, except for the number of timeslices.

Example 3.1.2. *We get back to the Student example. Students have grades throughout the scholarship. Students' Intelligence also evolves over time. These reports mean that random variables I and G evolve over time. They are time-dependent.*

Let be $t \in [\![0, T]\!]$ a discretized view of time. We suppose that every variable that composes the Bayesian network of the Student example is indexed with time. We introduce the corresponding dynamic Bayesian network $\mathcal{DBN}_{Student}$, that models the Student example.

$\mathcal{DBN}_{Student}$ is composed of the set of variables $\{(D^t, I^t, S^t, G^t, L^t) \mid t \in [\![0, T]\!]\}$, and its DAG is represented in Figure 3.4

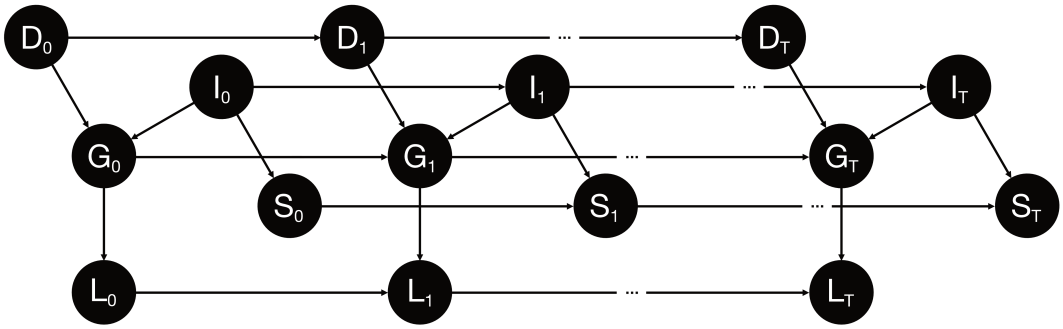


Figure 3.4: DAG structure of the Dynamic Bayesian network that can represent the Student example.

Hypotheses in DBNs

DBNs are supposed to be Markov dynamic systems, which means they follow the Markov assumption.

Definition 3.1.20 (Markov assumption)**.** *A dynamic system modeled with the set of variables* $\mathbf{X} = \left\{\mathbf{X}^{(t)}\right\}_{t\in[\![0,T]\!]}$ *satisfies the Markov assumption if, for all* $t \geq 0$*,*

$$\forall i \in [\![0, T-1]\!],\ \mathbf{X}^{(t+1)} \perp\!\!\!\perp \mathbf{X}^{(i)} \mid \mathbf{X}^{(t)} \tag{3.11}$$

$\mathbf{X}^{(t+1)}$ is conditionally independent of all the states before $\mathbf{X}^{(t)}$ given $\mathbf{X}^{(t)}$. It implies that the state at time $t+1$ only depends on the state at time t if known. In simple words, the Markov assumption assesses that the future is independent of the past given the present. Moreover, DBNs are supposed to be stationary.

Definition 3.1.21 (Stationary Markovian system)**.** *A stationary Markovian system if* $P(\mathbf{X}^{(t+1)} \mid \mathbf{X}^{(t)})$ *is time-invariant. Then, the distribution is fully represented by the time-independent transition model* $P(\mathbf{X}' \mid \mathbf{X})$ *such that*

$$\forall t \geq 0,\ P(\mathbf{X}^{(t+1)} = \mathbf{x}' \mid \mathbf{X}^{(t)} = \mathbf{x}) = P(\mathbf{X}' = \mathbf{x}' \mid \mathbf{X} = \mathbf{x}) \tag{3.12}$$

Because they are stationary Markov systems, DBNs can be summed up by two simple Bayesian networks. This provides a more formal definition of DBNs.

Definition 3.1.22. *A dynamic Bayesian network* $\mathcal{B} = \langle \mathcal{B}_0, \mathcal{B}_\rightarrow \rangle$ *is a set of two Bayesian networks* $\mathcal{B}_0$ *and* $\mathcal{B}_\rightarrow$*.*

- $\mathcal{B}_0$ *is the initial Bayesian network, it is the Bayesian network over* $\mathbf{X}^{(\mathbf{0})}$ *which represents the initial distribution over states;*
- $\mathcal{B}_\rightarrow$ *is a Two-Timeslices Bayesian Network (2-TBN) for a process over* $\mathbf{X}$*, it is the*

conditional Bayesian network over $\mathbf{X}'$ *given a subset of* $\mathbf{X}$ *that interfaces with* $\mathbf{X}'$.

The 2-TBN is then composed of inter-timeslice and intra-timeslice edges. Inter-timeslice edges are the arcs between timeslices. Because of the Markovian assumption, they correspond to the interactions between $\mathbf{X}$ and $\mathbf{X}'$. Intra-timeslice edges are the connections between variables in the same timeslice. They are uniquely composed of arcs between the nodes of $\mathbf{X}'$. This is a modeling constraint of DBNs.

The Definition 3.1.22 is equivalent to the description provided earlier in this section. The former is called the unrolled description of the DBN, while the latter is called the two-time-slice representation. The structure and CPDs of $\mathbf{X}^{(0)}$ of the unrolled DBN are the same as those for $(X_i)_i$ in $\mathcal{B}_0$, and, the structure and CPDs of $\mathbf{X}^{(t)}$ for $t > 0$ are the same as those for $(X'_i)_i$ in $\mathcal{B}_{\rightarrow}$. We can note that the 2-TBN representation is much more compact than the unrolled one. It is a compact representation, as there is no limit to how many Bayesian networks the DBN framework can generate. Moreover, it expresses the template nature of DBNs. Trajectories of different lengths can be studied from the same model.

Usual examples of DBNs

The first example of a dynamic Bayesian network that comes to mind is the Hidden Markov Model (HMM). An HMM is composed of a single state variable S and a single observation variable O at each timeslice t. The state variable S is a latent variable, as its value cannot be observed. As HMMs are Markov networks (PGMs with an undirected graph structure), they can be represented by a DBN. The variables of the corresponding DBN are $\{(S_t, O_t) \mid t \in [\![0, T]\!]\}$. Its DAG is represented in Figure 3.5. More complex structures of HMMs, such as factorial HMM or coupled HMM, are usually used to model complex systems.

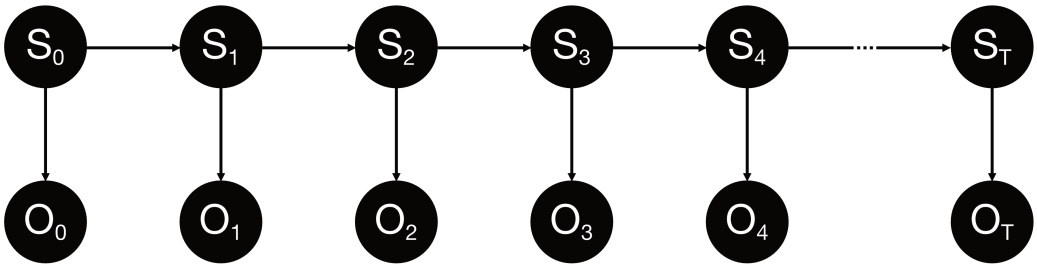


Figure 3.5: DBN representation of a Hidden Markov Model, composed of variables $\{(S_i, O_i) \mid t \in [\![0, T]\!]\}$ such that S_is represent the state variables, and O_is are the observable variables.

It is possible to consider a non-stationary DBN. However, the structure of such DBN will not allow describing the DBN with the set $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$. The benefits of such a structure for learning would then be lost.

3.2 Inference in Bayesian networks

Inference in Bayesian Networks (BNs) is the process of using a set of observations to make predictions about the values of unobserved variables in the system. BNs provide a powerful framework for reasoning about uncertain knowledge representation, and the principles of inference in the probability theory can be applied to make predictions based on incomplete or noisy data.

After exposing the principles of inference, we detail the state-of-the-art techniques for inference in BNs. Exact inference techniques for BNs can compute the exact posterior distribution over a set of variables. Still, they can be computationally expensive and may not be feasible for large systems. Optimization-based inference techniques can be used to find the Maximum A Posteriori (MAP) estimate of the variables, which is a computationally efficient way to make predictions. Particle-based inference techniques, such as particle filtering, can approximate the posterior distribution using a set of particles, which can provide a more accurate estimate of the posterior distribution in some cases. This section introduces these different inference techniques for Bayesian networks and discusses their strengths and weaknesses, particularly with scarce data.

3.2.1 Reasoning on uncertain knowledge representation

The main purpose of knowledge representation is knowledge reasoning. The previous section has introduced BNs as a framework for Knowledge Representation and Reasoning (KRR). Now, we wonder how we can interrogate the model to answer questions on the system. These questions can be of two natures. Deductive questions consist of interrogations on the logical structure of relationships between concepts, and inductive questions concern interrogations on concept states with particular evidence.

The notion of inference

Supposing we represent the state of the knowledge representation with random variables X_i, evidence on the system is a set of events in the form $\{X_i = x_i \mid i \in [\![1, N]\!]\}$. The task of interrogating the model to answer questions is called inference.

Definition 3.2.1 (Inference)**.** *The task of inference is the process of reaching answers to questions on the basis of evidence and reasoning.*

The notion of inference was first provided in expert systems by inference engines that answered queries with automated logical inference thanks to the First-Order Logic (FOL) structure of KRR [KS89]. The task of inference has been extended to the probability theory. We detail the tools employed to formalize inference requests with probability theory.

Probability queries

Basic requests for knowledge reasoning are probability queries. Probability queries are requests on the state of the system, expressed with conditional probabilities. It computes the posterior probability distribution over the values of a set of variables, given that other variables have known values. They are composed of two entities.

- Evidence $\mathbf{e}$ corresponds to the information we have on the system.
- The query variables $\mathbf{Y}$ correspond to the variables we want to evaluate, given the evidence.

Definition 3.2.2 (Evidence)**.** *Evidence is a set of information or observations that can be used to update or confirm the belief about a certain state.*

Definition 3.2.3 (Probability query)**.** *The probability query of random variables* $\mathbf{Y} \subseteq \mathbf{X}$ *given* $\mathbf{E} = \mathbf{e}$*, with* $\mathbf{E} \subseteq \mathbf{X} \setminus \mathbf{Y}$ *is the probability* $P(\mathbf{Y} \mid \mathbf{E} = \mathbf{e})$*.*

Probability queries are the basic tools for the task of inference. When $\mathbf{Y} = \mathbf{X}$ and $\mathbf{E} = \varnothing$, the probability query corresponds to the joint distribution over $\mathbf{X}$. When the set of query variables is strictly included in the non-evidence variables, i.e., $\mathbf{Y} \subset \mathbf{X} \setminus \mathbf{E}$, the probability query consists in computing the marginal of $\mathbf{Y}$ in the distribution that states $\mathbf{E} = \mathbf{e}$.

Maximum A Posteriori queries

The Maximum A Posteriori (MAP) query is a major application of inference. MAP queries are searches for the most likely assignment to all non-evidence variables. They interrogate the value of all the variables of the model. Sometimes, the MAP should be computed on a restricted set of variables like the query variables for probability queries.

Definition 3.2.4 (Maximum A Posteriori query)**.** *The Maximum A Posteriori query of random variables* $\mathbf{X}$ *given* $\mathbf{E} = \mathbf{e}$*, with* $\mathbf{E} \subseteq \mathbf{X}$ *is*

$$MAP(\mathbf{W} \mid \mathbf{e}) = \arg\max_{\mathbf{w} \in \mathbf{W}} P(\mathbf{w} \mid \mathbf{e}) = \arg\max_{\mathbf{w} \in \mathbf{W}} P(\mathbf{w}, \mathbf{e}) \tag{3.13}$$

with $\mathbf{W} = \mathbf{X} \setminus \mathbf{E}$*.*

Definition 3.2.5 (Marginal Maximum A Posteriori query)**.** *The marginal Maximum A Posteriori query of random variables* $\mathbf{Y} \subset \mathbf{X}$ *given* $\mathbf{E} = \mathbf{e}$*, with* $\mathbf{E} \subseteq \mathbf{X} \setminus \mathbf{Y}$*, is defined by Equation 3.14*

$$MAP(\mathbf{Y} \mid \mathbf{e}) = \arg\max_{\mathbf{y} \in \mathbf{Y}} P(\mathbf{y}, \mathbf{e}) = \arg\max_{\mathbf{y} \in \mathbf{Y}} \sum_{\mathbf{z} \in \mathbf{Z}} P(\mathbf{y}, \mathbf{z}, \mathbf{e}) \tag{3.14}$$

with $\mathbf{Z} = \mathbf{X} \setminus (\mathbf{E} \cup \mathbf{Y})$*, the set of marginalized variables.*

Insights into the complexity of inference

Because the state space of the query variables can be extensive, the inference task is challenging in practice. Basically, the straightforward approach to inference is a brute-force computation. For instance, the computation of the probabilistic probability query of a set of query variables $\mathbf{Y}$ can be expressed as Equation 3.15.

$$P(\mathbf{Y}) = \sum_{\mathbf{y} \in \mathbf{Y}} P(\mathbf{y}) = \sum_{y_1 \in Y_1} ... \sum_{y_N \in Y_N} P(y_1, ..., y_N) \tag{3.15}$$

According to the definition of conditional probabilities, any probability query, even with evidence, can be written with Equation 3.15. Inference implies the generation of a joint distribution that blows up exponentially. Therefore, the task of inference is NP-hard and probably requires exponential time in the worst case.

To tackle the intractability of the inference task, some instances of knowledge representation, such as Probabilistic Graphical Models (PGMs), have been designed to reduce the complexity of inference. When inference is not tractable, we can still perform approximate inference methods. We recall the purpose of inference in Bayesian Networks (BNs). It aims at answering the probability queries, defined in Definition 3.2.3, with subsets of the BN variables as query and evidence variables. Bayesian inference entirely relies on the BN Conditional Probability Distributions (CPDs). In a nutshell, it consists of computing probabilities from evidence on known events. The particularity of inference in BN is that inference benefits from the CPDs that describe the distribution using the chain rule and the Bayes theorem to answer probability queries.

Definition 3.2.6 (Bayes theorem)**.** *Let A and B be two events such that $P(B) \neq 0$. The Bayes theorem states that*

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{3.16}$$

We can see the Bayes theorem as a Bayesian update of the information of the distribution.

$$P(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{E} = \mathbf{e} \mid \mathbf{Y})P(\mathbf{Y})}{P(\mathbf{E} = \mathbf{e})} \tag{3.17}$$

The prior belief on the $\mathbf{Y}$ distribution is considered in the right term of Equation 3.17 through $P(\mathbf{Y})$. In contrast, the left term represents the posterior distribution of $\mathbf{Y}$ with $P(\mathbf{Y} \mid \mathbf{E})$.

3.2.2 Exact inference

Exact inference is inference where the probability query is computed exactly. The computations induced by exact inference can be extremely resource-consuming. The complexity

of these computations may vary depending on the structure of the studied Bayesian network. The BN structure indeed implies the factorization of the probability distribution, and this factorization is observed in the writing of probability queries. We only provide the main ideas that motivate the different algorithms of exact inference and their algorithmic implementation.

Variable elimination

Suppose a Bayesian network with a chain structure. The probability roughly propagates through the chain thanks to the chain rule, which states a marginal probability can be rewritten as a product – see Equation 3.10. Factorization implies factors that are recursively included in each other. This is the main principle of the Variable Elimination (VE) algorithm. This rewriting can also be done on tree-structured Bayesian networks and adapted to work for DAG with a polytree structure. However, the presence of a cycle in the Bayesian network makes impossible the exact inference with propagation. The main idea behind the VE algorithm is to reduce the problem of exact inference in a random BN to exact inference in a tree-structured Bayesian network. It uses factor graphs to operate this simplification.

The Hammersley–Clifford theorem shows that PGMs, such as Bayesian and Markov networks, can be represented as factor graphs. The VE algorithm uses the factors contained in the factor graph to iteratively marginalize out variables from the distribution. It supposes a given ordering eliciting the order of variables for elimination. For each variable given by the ordering, the principle of the VE algorithm is to multiply all the factors containing this variable to marginalize it from the product factor. This marginalization leads to a new factor employed to replace the factors containing the marginalized variable.

The VE algorithm is designed to answer marginal probability queries for which evidence $\mathbf{E}$ is supposed to be empty. The computation of conditional probability queries $P(\mathbf{Y} \mid \mathbf{E} = \mathbf{e})$ can also be computed with the VE algorithm, as conditional probabilities can be rewritten as a quotient of marginals, as reported in Equation 3.18.

$$P(\mathbf{Y} \mid \mathbf{E} = \mathbf{e}) = \frac{P(\mathbf{Y}, \mathbf{E} = \mathbf{e})}{P(\mathbf{E} = \mathbf{e})} \tag{3.18}$$

That is, we can compute the conditional probability by performing variable elimination on $P(\mathbf{Y}, \mathbf{E} = \mathbf{e})$ and then on $P(\mathbf{E} = \mathbf{e})$. When the assignments for variables $\mathbf{E}$ are fixed, then the values of factors on subsets of $\mathbf{E}$ are set to the ones specified by $\mathbf{e}$.

Still, the VE algorithm has two major drawbacks. First, finding the optimal ordering is NP-hard in the worst case [Coo90]. Second, two different queries need two runnings of the algorithm, even if they rely on the same set of variables [KF09].

Sum-product message-passing and junction trees

The VE algorithm can be interpreted as the propagation of a message when the BN is tree-structured. This is called a message-passing process. The factor computed from marginalizing a variable would be the message summarizing all the information contained in the descendants of this variable. Messages do not change, even when the query is changing. Computing them over and over leads to computational waste. To tackle this issue, the sum-product message-passing algorithm has been introduced.

The main idea of the sum-product message-passing algorithm is to store these messages, not to compute them twice. Once all messages are computed, any marginal probability can be computed in constant time from the product of messages from neighbor variables [Pea88]. Still, the sum-product message-passing algorithm only works when the graph structure of the Bayesian network is a tree. Therefore, one introduces junction trees to transform the structure of Bayesian networks that are not tree-structured into a tree of variable clusters.

Junction trees are the source of another algorithm for exact inference in Bayesian networks. The junction tree algorithm generalizes the message-passing approach to graph structures more complex than tree structures. In broad terms, the junction tree algorithm consists in the form of message-passing on the junction tree of the Bayesian network. It defines a potential for each clique that is the product of all factors assigned to it and computes messages from them [MJ99]. However, it has a running time that is potentially exponential in the size of the largest cluster. Finding the optimal tree is as computationally expensive as finding the optimal ordering in variable elimination. Overall, exact inference techniques have strong limitations in terms of applicability.

3.2.3 Optimization-based inference

Other kinds of inference have been developed to perform inference when the exact inference is not tractable. Among them are optimization-based inference techniques. The principle of optimization-based inference is principally to define a target class of convenient and tractable distributions and to search for an instance of this class that is the best fit for the target distribution. The queries are then performed on the chosen simpler distribution. One can also refer to variational inference methods for optimization-based inference. Köller et al. assert that optimization-based inference can be divided into three categories [KF09]. Optimization-based inference algorithms use strong mathematical tools that are out of the scope of this thesis. Still, we present the main methods and the ideas behind them.

Comparing exact and approximate inference techniques

To measure and compare the differences between the target distribution p and an approximating distribution q, we use the notion of Kullback-Leibler (KL) divergence. The KL divergence measures differences in the information contained within two distributions.

Definition 3.2.7 (Kullback-Leibler divergence)**.** *The Kullback-Leibler divergence between two distributions q and p with discrete support is defined as*

$$KL(q\|p) = \sum_x q(x) \log \frac{q(x)}{p(x)} \tag{3.19}$$

Loopy-Belief Propagation

We saw in Section 3.2.2 that exact inference could be performed either in a tree-structured Bayesian network or in a Bayesian network with a non-tree structure by considering the junction tree instead of the initial structure of the BN. However, the complexity of the junction tree algorithm still grows exponentially in the size of the largest cluster, so it is not possible to apply the junction tree on some too-complex Bayesian networks. Instead of using a single pass of message-passing, the Loopy-Belief Propagation (LBP) algorithm iterates over edges for message-passing. The LBP algorithm is a kind of optimization-based inference algorithm that implies clique-tree message-passing schemes on non-tree structures [MWJ99]. Rather than using the junction tree, LBP consists of a general message-passing algorithm in a general-purpose cluster graph. The LBP algorithm ignores the cycles in the graph, supposing the convergence will happen with enough passages.

Expectation Propagation

Another optimization-based inference method is the Expectation Propagation algorithm. While clique-tree message passing schemes on non-tree structures rely on approximating the structure and passing exact messages, the Expectation Propagation algorithm consists of message propagation on clique trees with approximate messages [Min01]. Among the main optimization-based inference techniques is also the mean-field inference. The purpose of mean-field inference is to consider that the approximating distribution q of the target distribution can be factorized into a product of factors representing the target distribution on each variable. It then becomes easier to optimize the KL divergence by coordinate descent over the variables.

Variational approaches

The optimization-based inference techniques detailed above can be analyzed variationally. They can indeed be described as variational optimization strategies for the energy functional, the negative equivalent to the Helmholtz free energy from statistical physics [IKU08]. Optimization-based inference aims to represent probability queries as entropy minimization problems. Thus, other optimization-based inference techniques can be derived from classic variational approaches [ZBKM18], such as convex duality [SHPU12].

3.2.4 Particle-based inference

The particle-based inference is another class of approximate inference. The principle of particle-based methods is to approximate the target probability distribution as a set of instantiations to the network variables, called particles. The provided particles give a good approximation of the target distribution. Indeed, the estimator of a function $\mathcal{F}$ on the distribution P of a Bayesian network verifies Equation 3.20. It can be expressed as the expectation of the function $\mathcal{F}$ applied to the N generated particles from P. The greater the number of generated particles N, the better the estimator.

$$E_P(\mathcal{F}) = \frac{1}{N}\sum_{i=1}^{N}\left[\mathcal{F}(X^{(i)})\right] \text{ with } X^{(i)} \text{ is the } i^{\text{th}} \text{ sample from } P \tag{3.20}$$

Forward sampling, rejection sampling, and importance sampling

The most straightforward particle-based method is forward sampling. Forward sampling generates particles by following the topological order of the Bayesian network [Hen88]. The samples generated with forward sampling are then studied according to Monte-Carlo estimation. Algorithm 3.1 details the implementation of forward sampling.

Algorithm 3.1: Forward sampling algorithm in a Bayesian network $\mathcal{B}$

Data: $\mathcal{B}$
Result: $\mathbf{x} = (x_1, ..., x_n)$
Let $X_1, ..., X_n$ ordered in a topological ordering of the variables of $\mathcal{B}$
for $i \in [\![1, n]\!]$ **do**
  $\mathbf{u}_i \leftarrow \mathbf{x}_{|\mathbf{Pa}_{X_i}}$
  Sample x_i from $P(X_i \mid \mathbf{Pa}_{X_i} = \mathbf{u}_i)$
end

When the probability query concerns conditional probability, the forward sampling may not be accurate as the marginal probability of the evidence could be very low. The forward sampling procedure for conditional probability queries is to produce samples and reject the ones that do not fit with the evidence state. It is called rejection sampling [Cow98]. A low marginal probability $P(\mathbf{e})$ induces large rejected particles (with N samples generated with forward sampling, the expected number of non-rejected samples is $N \times P(\mathbf{e})$). Importance sampling tackles this issue. Importance sampling consists in sampling a new distribution which is defined as a function of the joint distribution [SP89]. Then, it normalizes the samples to keep the relation between the Monte-Carlo estimator of the samples and the target distribution.

Markov Chain Monte Carlo approaches

Markov Chain Monte-Carlo (MCMC) methods are other techniques of particle-based inference. They consist of a Markov chain applied to assignments to all the variables in the BN [GRS95]. The convergence of MCMC methods does not depend on the smallness of $P(\mathbf{e})$. In detail, the Markov chain relies on the state of the model variables $\mathbf{X}$. It defines a sequence of variables $\mathbf{x}_1, ..., \mathbf{x}_n$ by initializing the state of $\mathbf{X}$ according to a probability $P^{(0)}(\mathbf{X})$ and computing the next state from the transition probability distribution $P(\mathbf{x}_i \mid \mathbf{x}_{i-1})$. Markov chain follows the Markov assumption. The transition probability distribution does not vary with i.

A converging MCMC method has a stationary distribution that equals the target probability distribution. It first runs the Markov chain from the initial state of the system for a burning period that it rejects. Then, it collects the samples generated by the Markov chain run for a fixed number of steps to compute the Monte Carlo estimator. The purpose of the burning period is to reject the generated samples that are not mixed yet. At the beginning of the Markov chain, successive states still depend on each other. Considering them in the Monte Carlo estimator would alter its convergence. The general formulation of MCMC is given in Algorithm 3.2.

Algorithm 3.2: Markov Chain Monte-Carlo (MCMC)

Data: Initial state distribution $P^{(0)}(\mathbf{X})$, Markov chain transition model $\mathcal{T}$, Number of samples N_{samples}
Result: $\mathbf{x}^{(0)}, ..., \mathbf{x}^{(N_{\text{samples}})}$

Sample $\mathbf{x}^{(0)}$ from $P^{(0)}(\mathbf{X})$
for $r \in [\![1, N_{samples}]\!]$ **do**
    Sample $\mathbf{x}^{(r)}$ from $\mathcal{T}(\mathbf{x}^{(r-1)} \to \mathbf{X})$
end

The most prevalent MCMC technique is the Metropolis-Hastings (MH) algorithm [CG95]. The MH algorithm is also the most versatile MCMC method. It relies on two elements: a transition kernel that must be specified and an acceptance probability defined from the transition kernel and the joint distribution.

However, in the context of a Bayesian network, the Gibbs sampling is more convenient, as the state of a Bayesian network can be decomposed in multiple variables $(X_1, ..., X_N)$ with known conditional probability distributions $\forall i \in [\![1, N]\!], P(X_i \mid X_{-i})$ [Hry90]. The Gibbs sampling algorithm iteratively updates the state of each variable thanks to sampling on conditional probabilities [CG92]. It is detailed in Algorithm 3.3.

Gibbs sampling corresponds to the MH algorithm with a transition kernel that alternates between the $\{P(x'_i \mid x_{-i})\}_i$ and an acceptance probability equal to one. Gibbs sampling is a

Algorithm 3.3: Gibbs sampling algorithm in a Bayesian network $\mathcal{B}$

Data: $\mathcal{B}$, number of Gibbs iterations N_{Gibbs}, burn-in period M, sampling period P_S
Result: $\mathbf{x}$

Let $X_1, ..., X_n$ be an ordering of the variables of $\mathcal{B}$.
Initialize $\mathbf{x}^{(0)}$ randomly.
for $r = 1$ *to* N_{Gibbs} **do**
  for $j = 1$ *to* k **do**
    MB $\leftarrow$ GetMarkovBlanket(X_j)
    $\mathbf{e} \leftarrow (X_1^{(r)}, ..., X_{j-1}^{(r)}, X_{j+1}^{(r-1)}, ..., X_N^{(r-1)})_{|\text{MB}}$
    Sample $x_j^{(r)}$ from $P(X_j \mid \mathbf{e})$ ▷ with exact inference
  end
  $\mathbf{x}^{(r)} \leftarrow (x_1^{(r)}, ..., x_k^{(r)})$
  if $r > M$ **then**
    if $r \equiv 0[P_S]$ **then**
      Add $\mathbf{x}^{(r)}$ to $\mathbf{x}$
    end
  end
end

straightforward strategy for approximate inference in Bayesian networks. We will present more details on Gibbs sampling implementation for specific instances of Bayesian networks in Chapter 4.

Convergence of MCMC methods

The main interest of the particle-based approximation methods compared to other approximation methods is that true posterior distribution is theoretically reachable. Unfortunately, these particle-based methods have several drawbacks. First, their convergence is not straightforward. For instance, Geman and Geman have shown that Gibbs sampling from a strictly positive distribution converges [GG84].

Definition 3.2.8 (Strictly positive distribution)**.** *A strictly positive distribution P has values $P(x) > 0$ for all x.*

Property 3.2.1 (Convergence of Gibbs sampling from a strictly positive distribution)**.** *Let be $\mathbf{x}$ a set of samples. If the sampled probability distribution is strictly positive, then for*

every function F such that $E[F(x)] < \infty$, we have:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{l=1}^{N} F(\mathbf{x}(t_l)) = E[F(\mathbf{x})] \tag{3.21}$$

York proves that the irreducibility of the Markovian process relative to the Bayesian network is sufficient for Equation 3.21 to be verified [Yor92]. Then, Gibbs sampling could be used to generate samples approximating the joint distribution of the BN variables $(x_i)_{i\in[1,n]}$, or any marginal distributions on a subset of variables.

Definition 3.2.9 (Irreducible Markovian process)**.** *An irreducible Markovian process is a Markovian process such that for all states* $\mathbf{x}$ *and* $\mathbf{x}'$ *such that* $f(\mathbf{x}) > 0$ *and* $f(\mathbf{x}') > 0$*, there exists a strictly positive probability such that the Markovian process can reach* $\mathbf{x}'$ *from* $\mathbf{x}$ *in a finite number of iterations of the Markov chain.*

Nevertheless, tailoring MCMC methods to get a Bayesian network to these conditions may be tricky. The Gibbs sampling convergence in reducible MCMC will be extensively discussed in Chapter 5.

Once the issue of reducibility is solved, Markov chain approaches are guaranteed to converge to a globally optimal solution. Still, the time it needs to converge is not a naive question. In practice, convergence time is hard to quantify: finding a good solution in finite time is not ensured. Köller et al. suggest a hybrid strategy to use Markov chains in practice: they recommend using multiple chains in parallel instead of a single chain and define metrics to evaluate their convergence [KF09].

Particle-based approaches still provide relatively tractable approximate inference methods. In particular, when data is scarce, exact inference techniques are not viable, and optimization-based approaches may require too many assumptions. Then, MCMC techniques are promising tools for inference from very limited evidence.

3.3 Learning Bayesian networks

Bayesian networks provide a powerful framework for knowledge-enhanced machine learning, and data is a crucial component of this framework. In order to use Bayesian networks to make predictions about real-world systems, we can learn the structure and parameters of the network from the data. Parameter learning in Bayesian networks is the process of estimating the Conditional Probability Tables (CPTs) that specify the relationships between the variables in the network. This can be done using fully-observed data, where all of the variables in the network are observed, or using missing data, where some of the variables are unobserved.

In this section, we introduce the basics of parameter learning in Bayesian networks and discuss the different approaches that can be used to learn the parameters from data. This section points out the challenges associated with the parameter learning techniques, such as the curse of dimensionality, that will be the main issue when performing parameter learning in our framework models.

3.3.1 Integrating data-driven techniques into knowledge representation

Probabilistic Graphical Models (PGMs) provide a framework for Knowledge Representation and Reasoning (KRR). We saw in Section 3.1.2 that the structure of the knowledge representation must be known to address knowledge reasoning. Data-driven techniques can be used to enhance the effectiveness and efficiency of probabilistic graphical models in a number of ways. One approach is to use machine learning algorithms to learn the model parameters from data.

PGMs before the rise of big data

First, the structure of a PGM was supposed to be an input to knowledge reasoning constructed by hand. This task was given to domain experts most of the time. Nevertheless, representing knowledge is a hard task. It encompasses the specification of both the qualitative and the quantitative structure. For instance, for a Bayesian network, it means determining the structure of the DAG and the values in the CPDs. Constructing the structure of a PGM of reasonable size requires a massive effort by qualified people. Köller et al. give the order of magnitude of a few weeks or months for the time required to determine the structure of a large network [KF09]. It could be hard to interrogate these qualified people. Worse, they could not exist, as some systems have no experts with sufficient knowledge to specify the distribution that rules it. Also, the distribution may change over time. The effort of designing the knowledge representation with humans is not viable if it must be reproduced on a regular basis.

The interests of performing machine learning on KRR techniques

However, with the major advances in data-driven machine learning, the use of knowledge representation is questionable. We can wonder how KRR techniques are relevant, while at the same time, data-driven machine learning can produce a structure that would answer the same question with fewer efforts. Still, knowledge representation and reasoning methods keep some of their benefits compared to data-driven machine learning. Brachman et al. claim that KRR-based systems are easy to debug, modify, and explain [BL04]. In the context of modeling complex systems, these benefits are essential. As precise as the decisions taken from data-driven machine learning techniques can be, most of them are black boxes, which can be prejudicial during use.

Moreover, learning a KRR-based model would technically be quicker and require fewer data than data-driven machine learning techniques, such as deep learning algorithms. Cozman et al. list ten reasons why KRR may enhance data-driven machine learning [CM21] that can be summarized in three categories. First, including the KRR techniques in data-driven machine learning would allow the integration of biases in the learning task to fasten it (e.g., for the cold-start problem) or prevent unwanted behavior (such as discriminatory ones). If biases are non-granted in artificial intelligence tasks, those included in KRR techniques are supposed to correct biases already in the data. Second, it grants a better understanding of the algorithm operations. Indeed, from the system designer's point of view, it is a way to turn artificial intelligence easy to debug. Modifications on the model are easier as the knowledge representation (e.g., a DAG) is fully intelligible. From the system users' point of view, it increases the comprehension of the modeled system. Finally, because KRR techniques require fewer data, it means that the energy consumption of knowledge-enhanced machine learning would be lower than data-driven approaches.

The learning task for a Bayesian network

Bayesian networks can either be seen as a KRR technique or a machine learning model. The former use of BN was representational. The parameters and the structure of the BN were supposed to be known. However, the increased amount of data has brought BN to machine learning to fit the available data better.

The model learning of a Bayesian network consists of two different tasks: parameter learning and structure learning [DSA11]. On the one hand, learning parameters of a Bayesian network consists in learning the values in the CPDs of the model, supposing that the structure is known. On the other hand, structure learning corresponds to the search for the best structure of the DAG on which the Bayesian network relies.

In this work, we restrict model learning to parameter learning. The structure learning task broadly consists of an iterative version of parameters learning on multiple DAG structures. We will see in the following that the computations induced by parameter learning can already

be hard to compute. Including these calculations as subroutines would ineluctably lead to an intractable process. Therefore, we suppose that the structure of the DAG is known and that it will not change over the parameters learning task.

Quick look at deeper inclusions of knowledge in machine learning techniques

Achieving BN model learning is an important inclusion of machine learning techniques in knowledge representation, but other techniques go even deeper. This quick overview is a footpath into another research domain called Statistical Relational Learning (SRL). SRL regroups algorithms that use machine learning techniques for knowledge reasoning on systems whose knowledge is composed of a complex relational structure. We roughly decompose SRL into two categories.

On the one hand, we present SRL approaches based on logical programs. The main model in this category is Inductive Logic Programming (ILP). ILP consists of logic programs extracted from a set of positive and negative examples, given an encoding of the knowledge of the system [Mug91]. The knowledge of the system is encoded using Horn clauses. It entails the KRR basis of ILP. The positive and negative examples in a logical database of facts and the logic programs generated by ILP are hypotheses on the knowledge that are inductively inferred from positive and negative examples. This is the machine learning part of this category of programs. ILP has been extended to probability theory with Probabilistic Inductive Logic Programming (PILP) [DRK08]. In PILP, the three kinds of uncertainty in KRR reported in Section 3.1.1 are considered thanks to the probability theory. Other logical-based SRLs, such as Bayesian Logic Programs, fully describe the knowledge representation from a set of textual relations between the model variables and are then subject to machine learning from data.

The other approach for SRL is based on Probabilistic Relational Models (PRMs). PRMs use first-order logic to represent knowledge thanks to PGMs [GFK+07]. The naive approach to using data to enhance a PGM is to perform model learning on the PGM from the data. However, the knowledge representation emphasized by the structure of the PGM would be lost, as model learning would imply a new structure. The idea behind PRMs is to define a higher-order structure with logical relations that cannot change even with model learning. Then, PRMs are composed of constraints that summarize the knowledge representation and that will impact model learning. For example, Relational Bayesian networks (RBNs) are PRMs. A relational Bayesian network considers a set of relations between its variables and represents them with a Bayesian network [Jae01]. The main purpose of RBN is that the relations can be the same between different variables: at a high range, it defines classes of relations that are learned together. Other SRL techniques have been developed on other PGM than Bayesian networks, such as Markov Logic Networks, which use Markov networks [RD06].

Both logical and graphical-based SRL methods are designed to enhance the symbolic reasoning

in KRR with data-driven methods. For instance, Jaeger compares Graph Neural Networks, which are data-driven deep learning techniques, with RBNs [Jae22]. He deplores that the interpretability of neural networks is way more difficult to apprehend for a human than that of symbolic representations, even if computational performances of GNN are better than RBN ones. Inclusions of KRR into data-driven machine learning techniques are promising approaches for developing interpretable models of uncertainty-prone complex systems.

3.3.2 Parameter learning from fully-observed variables

Parameter learning is an essential task for Bayesian networks, as it is the first footpath to bring them from KRR techniques to machine learning models. First, we focus on the parameter learning task for the Bayesian network when the data are complete, that is to say when all the BN variables are fully observed. The data on the model give all the information on the BN variables. We denote $\mathcal{D}$, the data on the distribution represented by the BN.

Maximum likelihood estimate

The Maximum Likelihood Estimate (MLE) is the most common approach when all variables are observed. The MLE is a statistical estimator of the parameters: for each possible event, the estimated probability $\hat{P}$ of the event is computed from its appearing frequency in the database.

$$\hat{P}(X_i = x_k | \mathbf{Pa}_{X_i} = \mathbf{x}_j) = \hat{\theta}^{\text{MLE}}_{i,j,k} \text{ where } \hat{\theta}^{\text{MLE}}_{i,j,k} \text{ verifies } \frac{\partial \mathcal{L}(\mathcal{D}|\theta)}{\partial \theta_{i,j,k}} \left(\hat{\theta}^{\text{MLE}}_{i,j,k}\right) = 0 \tag{3.22}$$

In Bayesian networks, the MLE of parameters $\{\theta_{i,j,k}\}_{i,j,k}$ can be written as a quotient of counts of events, as reported in Property 3.3.1.

Property 3.3.1 (Maximum likelihood estimate of the distribution in a Bayesian network)**.** *In a Bayesian network, the maximum likelihood estimate of $\theta_{i,j,k}$ can be expressed as*

$$\hat{\theta}^{MLE}_{i,j,k} = \frac{N_{i,j,k}}{\sum_l N_{i,j,l}} \tag{3.23}$$

with $N_{i,j,k}$ the count of events in the database where $X_i = x_k$ and $\mathbf{Pa}_{X_i} = \mathbf{x}_j$.

The set of counts $\{N_{i,j,k}\}_{(i,j,k)}$ are also called the sufficient statistics of the Bayesian network, as these counts give all the information required on the parameters of the network.

Maximum A Posteriori

Another approach to parameter learning from complete datasets is Bayesian estimation. The principle of Bayesian estimation of the network parameters comes from the Bayes rule. It expresses the posterior distribution of parameters $P(\theta|\mathcal{D})$ as a function of the prior distribution of parameters $P(\theta)$, related with the Bayes theorem – see Definition 3.2.6.

$$P(\theta|\mathcal{D}) \propto P(\mathcal{D}|\theta)P(\theta) \tag{3.24}$$

Because the distribution describing the dataset $\mathcal{D}$ is a multinomial law, we assume the prior distribution is a Dirichlet distribution. We introduce Dirichlet coefficients $\alpha_{i,j,k}$ for each entry in the CPDs. The prior distribution is then:

$$P(\theta) \propto \prod_i \prod_j \prod_k \theta_{i,j,k}^{\alpha_{i,j,k}-1} \tag{3.25}$$

If the dataset is composed of fully-observed i.i.d. samples, then the posterior distribution is also a Dirichlet distribution [KF09].

$$P(\theta \mid \mathcal{D}) \propto \prod_i \prod_j \prod_k \theta_{i,j,k}^{N_{i,j,k}+\alpha_{i,j,k}-1} \tag{3.26}$$

The parameters of the Bayesian network can be computed from the Maximum A Posteriori (MAP) estimation of the posterior distribution described in Equation 3.26.

$$\hat{P}(X_i = x_k|\mathbf{Pa}_{X_i} = \mathbf{x}_j) = \hat{\theta}_{i,j,k}^{\text{MAP}} \text{ where } \hat{\theta}_{i,j,k}^{\text{MAP}} \text{ verifies } \frac{\partial \mathcal{P}(\theta|\mathcal{D})}{\partial \theta_{i,j,k}}\left(\hat{\theta}_{i,j,k}^{\text{MAP}}\right) = 0 \tag{3.27}$$

Such as the MLE approach, the MAP of $\theta_{i,j,k}$ can be written as a quotient of the counts of events and the Dirichlet coefficients, as reported in Property 3.3.2.

Property 3.3.2 (Maximum a posteriori of the parameter distribution in a Bayesian network)**.** *The maximum a posteriori of $\theta_{i,j,k}$ can be expressed as*

$$\hat{\theta}_{i,j,k}^{MAP} = \frac{N'_{i,j,k}}{\sum_l N'_{i,j,l}} \tag{3.28}$$

with $\forall i, j, k,\ N'_{i,j,k} = N_{i,j,k} + \alpha_{i,j,k} - 1$.

The MAP approach is equivalent to the MLE approach when the Dirichlet coefficients are all equal to one. In practice, the Dirichlet coefficients $\{\alpha_{i,j,k}\}_{i,j,k}$ can be interpreted as the prior belief an expert can give on parameters. The MLE approach then supposes no prior belief.

3.3.3 Parameter learning from missing data

Nevertheless, in the majority of situations, some data are missing. Missing data can relate to observable or hidden (or latent) variables. Missing data on observable variables would mean that the state of these variables could have been specified. On the contrary, hidden variables are never observable: data cannot be collected on these variables.

In practice, the missing data can be categorized into three different natures:

- The probability of missingness of Missing Completly At Random (MCAR) data does not depend on the dataset.
- The probability of missingness of Missing At Random (MAR) data does depend on the observed part of the dataset.
- The probability of missingness of Missing Not At Random (MNAR) data does depend on both the observed and the missing parts of the dataset.

Deletion techniques

Naturally, the available techniques for learning the parameters of a BN depend on the nature of missing data. When missing data are MAR or MCAR, all the information required for parameter learning is contained in the database $\mathcal{D}$. The usual approach in such cases is to return to the task of parameter learning on a complete dataset by deleting incomplete rows of data. These deletions are called direct deletions, as we remove any entry of the database that is not complete. Factorized deletion can also be employed. With this approach, for each CPD, factorized deletion techniques consider every entry in the database where all variables concerned by the CPD are observed. These techniques are said to be inference-free. They do not require inference in the network to be performed.

However, when data are MNAR, the information contained in the database is not sufficient to learn the parameters of the model from deletion. The MNAR property implies every row has a missing value. The number of exploitable rows from direct or factorized deletion techniques would be reduced to zero.

The Expectation-Maximization algorithm

When data are MNAR, the parameter learning task requires using inference techniques. The most popular algorithm using inference to perform parameter learning is the Expectation-Maximization (EM) algorithm. The EM algorithm was introduced by Dempster et al. [DLR77]. It is an iterative algorithm relying on two intuitive principles to find the model parameters that lead to a maximum likelihood. First, when the model parameters are known, it is possible to derive the state of every variable given the state of its adjacent variables. Conversely, when the state of every variable is known, then finding the parameters that maximize the likelihood is easy. These two principles, respectively called E-step and M-step, are applied successively in the EM algorithm. In the E-step, the state of the hidden BN variables is computed from the BN parameters and the state of observed BN variables. In the M-step, the BN parameters are computed from the distribution of all BN nodes. We detail the implementation of the EM algorithm, which alternates between the E-step and the M-step, in Algorithm 3.4.

We detail the principles of the EM algorithm, as it is one of the main algorithms employed in our thesis work. Let be f the probability density associated with the dataset $x = (y, z)$

Algorithm 3.4: Expectation-Maximization algorithm in Bayesian network $\mathcal{B}$

Data: Initial set of parameters $\boldsymbol{\theta}^0$, partially observed data $\mathcal{D}$, number of EM iterations N_{EM}
Result: $\boldsymbol{\theta}^t$

for $t = 0, ..., N_{EM}$ **do**
  % E-step
  for $i = 1, ..., N$ **do**
    for *each* $x_i, \mathbf{u}_i \in Val(X_i, Pa_{X_i}^{\mathcal{G}})$ **do**
      $\overline{M}[x_i, \mathbf{u}_i] \leftarrow 0$
    end
  end
  for $m = 1, ..., M$ **do**
    Run inference on $\langle \mathcal{G}, \boldsymbol{\theta}^t \rangle$ with evidence $\mathcal{D}_m$
    for $i = 1, ..., N$ **do**
      for *each* $x_i, \mathbf{u}_i \in Val(X_i, Pa_{X_i}^{\mathcal{G}})$ **do**
        $\overline{M}[x_i, \mathbf{u}_i] \leftarrow \overline{M}[x_i, \mathbf{u}_i] + P(x_i, \mathbf{u}_i \mid \mathcal{D}_m)$
      end
    end
  end
  % M-step
  for $i = 1, ..., N$ **do**
    for *each* $x_i, \mathbf{u}_i \in Val(X_i, Pa_{X_i}^{\mathcal{G}})$ **do**
      $\theta_{x_i|\mathbf{u}_i}^{t+1} \leftarrow \frac{\overline{M}[x_i, \mathbf{u}_i]}{\overline{M}[\mathbf{u}_i]}$
    end
  end
end

where y are data on observed variables and z are data on non-observed variables. Let be g the probability density that describes the observed data y with σ-finite measure dy. The objective of the EM algorithm is to find optimal parameters $\hat{\boldsymbol{\theta}}$ maximizing the likelihood $\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. The likelihood in the context of missing data is defined in Equation 3.29.

$$L(\boldsymbol{\theta}) = \log g(y|\boldsymbol{\theta}) = \log \int f(y, z|\boldsymbol{\theta}) dz \tag{3.29}$$

We introduce $k(z|y,\boldsymbol{\theta}) = f(x|\boldsymbol{\theta})/g(y|\boldsymbol{\theta})$, the conditional probability density of z given y according to a σ-finite measure dz. The mathematical idea behind the EM algorithm is to replace the maximization of the likelihood function of f with the successive maximization of conditional expectations $Q(\boldsymbol{\theta}^*, \boldsymbol{\theta})$ defined in Equation 3.30.

$$Q(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \int \log(f(x|\boldsymbol{\theta}^*))k(z|y,\boldsymbol{\theta})dz \tag{3.30}$$

At the r-th iteration of the EM algorithm, the two steps are then:

- E-step: computation of the expectation $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ where $\boldsymbol{\theta}^r$ are the model parameters at iteration r of the EM algorithm.
- M-step: update of the model parameters such that $\boldsymbol{\theta}^{r+1} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ thanks to the MAP parameter distribution, defined in Equation 3.28.

These two steps are repeated until convergence. The likelihood increases between two successive iterations of the EM algorithm: $L(\boldsymbol{\theta}^{r+1}) \geq L(\boldsymbol{\theta}^r)$ [Wu83]. Moreover, the likelihood will not change anymore once the convergence point of the EM algorithm is reached, as reported in Property 3.3.3.

Property 3.3.3 (Convergence of the EM algorithm)**.**

$$L(\boldsymbol{\theta}^{r+1}) = L(\boldsymbol{\theta}^r) \iff Q(\boldsymbol{\theta}^{r+1}, \boldsymbol{\theta}^r) = Q(\boldsymbol{\theta}^r, \boldsymbol{\theta}^r) \tag{3.31}$$

Stochastic approaches of the EM algorithm

Nevertheless, the computation of the expectation Q requires the exact inference of the joint posterior distribution. The computation of the conditional probability density k can be too complex to handle. In such circumstances, the E-step of the EM algorithm is not tractable. Stochastic versions of the EM algorithm have been developed to tackle the computational cost of the joint posterior calculations during the EM procedure [CCD95]. They rely on the sampling of the joint posterior distribution to perform the E-step.

Stochastic Expectation-Maximization (SEM) is a stochastic version of the EM algorithm that replaces the computation of the density $k(z|y,\boldsymbol{\theta})$ in the expectation $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ with the sampling of z^r of the non-observed variables according to the conditional probability density $k(z|y,\boldsymbol{\theta}^r)$ [CD88]. Then, it updates the model parameters from the pseudo-complete samples $x^r = (y, z^r)$.

Wei and Tanner have proposed a Monte-Carlo approach to the EM algorithm, called the Monte-Carlo Expectation Maximization (MCEM) algorithm [WT90]. Its implementation is given in Algorithm 3.5.

In practice, the MCEM algorithm corresponds to the EM algorithm with a Monte-Carlo

Algorithm 3.5: Monte-Carlo Expectation-Maximization algorithm in Bayesian network $\mathcal{B}$

Data: Initial set of parameters $\boldsymbol{\theta}^0$, partially observed data $\mathcal{D}$, number of EM iterations N_{EM}, number of samples N_{samples}

Result: $\boldsymbol{\theta}^t$

for $t = 0, ..., N_{EM}$ **do**
  % E-step
  for $i = 1, ..., N$ **do**
    for *each* $x_i, \mathbf{u}_i \in Val(X_i, Pa_{X_i}^{\mathcal{G}})$ **do**
      $\overline{M}[x_i, \mathbf{u}_i] \leftarrow 0$
    end
  end
  for $m = 1, ..., M$ **do**
    Initialize $P^{(0)}(\mathbf{X})$ randomly.
    $\mathbf{x}_m \leftarrow MCMC(P^{(0)}(\mathbf{X}), \mathcal{T}, N_{\text{samples}})$
    Add $\mathbf{x}_m$ to dataset $\mathcal{D}_{\text{MCMC}}$
  end
  % M-step
  for $i = 1, ..., N$ **do**
    for *each* $x_i, \mathbf{u}_i \in Val(X_i, Pa_{X_i}^{\mathcal{G}})$ **do**
      $\theta^{t+1}_{x_i|\mathbf{u}_i} \leftarrow \frac{\overline{M}[x_i, \mathbf{u}_i]}{\overline{M}[\mathbf{u}_i]}$
    end
  end
end

approximation of the E-step. The computation of the expectation $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ is replaced by an empirical expectation $Q_{r+1}(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ that is computed from m samples of z from the conditional probability density $k(z|y, \boldsymbol{\theta}^r)$. Then, at each iteration r, the MCEM algorithm consists of the following steps:

1. We generate i.i.d. samples $\{z^r(1), ..., z^r(m)\}$ from the conditional probability density $k(z|y, \boldsymbol{\theta}^r)$

2. We update the approximation $Q_{r+1}(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ of the expectation $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$ with

$$Q_{r+1}(\boldsymbol{\theta}, \boldsymbol{\theta}^r) = \frac{1}{m} \sum_{j=1}^{m} \log f(y, z^r(j)|\boldsymbol{\theta})$$

3. We update the model parameters with

$$\boldsymbol{\theta}^{r+1} = \arg\max_{\boldsymbol{\theta}} Q_{r+1}(\boldsymbol{\theta}, \boldsymbol{\theta}^r)$$

The MCEM algorithm is central to our thesis work. The model we have developed is prone to scarce data. Performing exact inference in this context is extremely challenging, as reported in Chapter 5. Consequently, the MCEM algorithm sounds like the right choice for parameter learning.

3.4 Discussion

In conclusion, Bayesian networks provide a powerful framework for knowledge-enhanced machine learning. They allow us to reason about uncertain knowledge representation and make predictions based on incomplete or noisy data. By using probability theory to represent uncertainty and by using PGMs to model the conditional dependencies between variables, we can create compact and intuitive models that can be used to make predictions in various real-world applications. We also highlighted BNs could be pushed further to include machine learning in knowledge representations more deeply.

In this chapter, we introduced the basics of Bayesian networks. We have seen how they can be employed for knowledge-enhanced machine learning applications. We focused on the different inference and parameter learning techniques that can be used to make predictions and estimate the model parameters from data. This chapter will serve as a useful introduction to Bayesian networks and provide the keys for developing a knowledge-enhanced machine learning framework for student modeling.

Chapter 4

E-PRISM, a framework for student modeling

In this chapter, we introduce the E-PRISM framework, which stands for Embedding Prerequisite Relationships In Student Modeling. E-PRISM is the core of our thesis work.

The motivation for developing E-PRISM is to provide a knowledge-enhanced machine learning framework that mobilizes symbolic and data-driven techniques for Intelligent Tutoring Systems (ITS). E-PRISM is designed to fit into a wide range of ITS. It aims to establish the connection between the data from learner traces and the models that can be extracted from the domain knowledge of experts. As a framework, it provides a structure for the elements of student modeling corresponding to a sort of overlay description. It supports the objectives of the ITS it is integrated into, such as diagnosing and predicting the learners' knowledge states. E-PRISM is an original contribution in its ability to embed prerequisite relationships within the learner model and exploit them for diagnosis and prediction.

Building trust and confidence in the models, especially regarding education, is essential for enabling teachers and learners to understand better how the models make recommendations and provide feedback. We have highlighted the interpretability gain of using Bayesian networks for knowledge-enhanced machine learning in Chapter 3. Consequently, ICI-based Bayesian networks, particular types of Bayesian networks, will be the basis for building the E-PRISM learner models. We specifically focus on how this can improve the interpretability of the models.

Moreover, one of the main contributions of the E-PRISM framework is the integration of prerequisite relations into the learner model. By incorporating these prerequisite relations into the student models, we show how E-PRISM can provide insights into the effective application of domain knowledge into learners' knowledge developed over time.

4.1 Components of the E-PRISM framework

First, we introduce the components of the E-PRISM framework. Specifically, we explain how E-PRISM modelizes domain and learners' knowledge in ITS. E-PRISM is inspired by an overlay learner model that would assume the learner's knowledge is a domain knowledge subset. We also describe the architecture of E-PRISM, which is designed to be modular and easy to implement in an ITS. The architecture allows for integrating a custom domain model, enabling flexible and customizable student modeling with the E-PRISM framework.

4.1.1 A framework for modeling student in ITS

E-PRISM implements two models as an overlay learner model, in which the learner model relates to a domain model. We present the details of the domain and learner models in E-PRISM. Notably, we point out how it is designed to support the integration of multiple data sources, such as student interactions with the ITS or performance on assessments. We discuss the constraints on its structure to ensure that it aligns with the description of overlay learner models provided in Chapter 2.

Domain model

In overlay learner modeling, the domain model is the basis for defining the learner model, as reported in Section 2.1.1. In the E-PRISM framework, the domain model entirely relies on the decomposition of the domain knowledge into Knowledge Components (KCs). We suppose the modeled KCs follow the definition given by Koedinger et al. in the Knowledge-Learning-Instruction (KLI) framework [KCP12], detailed in Definition 2.1.2. This is one of the ITS constraints implied by using the E-PRISM framework. The KCs in the E-PRISM domain model have a monitored granularity. We assume they can be attached to minimal tasks to assess their mastery. We will see in Section 4.1.2 that this is the source of another assumption on the relationship between KCs and instructional resources.

The domain model of the E-PRISM framework also contains information on the relationships between the KCs. We have seen in Chapter 2 that some ITS consider different KC relationships. In the E-PRISM framework, only the prerequisite relationships are considered. We define the deterministic prerequisite relationship in Definition 4.1.1.

Definition 4.1.1 (Deterministic prerequisite relationship)**.** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two knowledge components. There exists a deterministic prerequisite relationship from $\mathfrak{A}$ to $\mathfrak{B}$ if the mastery of $\mathfrak{B}$ implies the mastery of $\mathfrak{A}$ and the non-mastery of $\mathfrak{A}$ implies the non-mastery of $\mathfrak{B}$. $\mathfrak{A}$ is said to be the parent of $\mathfrak{B}$, and $\mathfrak{B}$ is said to be the child of $\mathfrak{A}$ in the sense of the prerequisite relationship.*

Definition 4.1.1 describes the prerequisite relationship as a deterministic order between knowledge components. Thus, the prerequisite relationships considered in the E-PRISM framework can be visualized as a directed symbolic graph. The vertices would be the KCs,

and there would be an edge for each prerequisite relationship from the prerequisite KC to the postrequisite KC. Such a graph draws a picture of the domain knowledge similar to the skill graph usually employed in Competence-based Knowledge Space Theory (CbKST) [HSHA06], which is a variant of the Knowledge Space Theory (KST) [DF12].

Notation 4.1.1 (Parents and children in the prerequisite structure)**.** *Let $\mathfrak{X}$ be a knowledge component. We denote $N_{parents}(\mathfrak{X})$ (resp. $N_{children}(\mathfrak{X})$) the number of $\mathfrak{X}$'s parents (resp. the number of $\mathfrak{X}$'s children), i.e., the number of $\mathfrak{X}$'s prerequisite KCs (resp. the number of $\mathfrak{X}$'s postrequisite KCs).*

- *The set of $\mathfrak{X}$'s parents is denoted $\mathfrak{Pa}_{\mathfrak{X}} = \{\mathfrak{Pa}_{\mathfrak{X},\mathfrak{i}} \,|\, i \in [1, N_{parents}(\mathfrak{X})]\}$;*
- *The set of $\mathfrak{X}$'s children is denoted $\mathfrak{Ch}_{\mathfrak{X}} = \{\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}} \,|\, i \in [1, N_{children}(\mathfrak{X})]\}$.*

Learner model

The proper learner model in overlay modeling traces the learner's knowledge by defining it regarding the domain model. In the E-PRISM framework, we suppose the learner's knowledge is a subset of the domain knowledge. Because the E-PRISM domain model consists of KCs, we define the learner's knowledge by assuming the learner's mastery of each KC is represented with a binary random variable. KCs can either be mastered or not mastered by students. In practice, the learner's knowledge is described with a set of mastered KCs and a set of not-mastered KCs. For a knowledge component $\mathfrak{X}$, we then introduce a binary random variable X which takes its values in $\{\neg x, +x\}$.

Notation 4.1.2 (Mastery events)**.** *For any knowledge component $\mathfrak{X}$, we denote "$X = +x$" the event "The knowledge component $\mathfrak{X}$ is mastered" and "$X = \neg x$" the event "The knowledge component $\mathfrak{X}$ is not mastered". X is the random variable associated with the mastery of the KC $\mathfrak{X}$.*

As explained in Chapter 3, uncertainty must be considered when studying the learner's knowledge. We then measure the learner's mastery of a particular KC $\mathfrak{X}$ with the marginal distribution $P(X)$. The learner's knowledge state is fully described by the distribution $P(\mathbf{X})$ where $\mathbf{X}$ is the vector of random variables associated with the mastery of every KCs.

Besides the uncertainty of the KC masteries, depicted with probability theory, we assume there is also uncertainty in the effect of the prerequisite relationships described in the domain model. Thus, we consider prerequisite relationships probabilistic rather than deterministic in the E-PRISM learner model. We define the probabilistic prerequisite relationship between two knowledge components in Definition 4.1.2.

Definition 4.1.2 (Probabilistic prerequisite relationship)**.** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be two knowledge components. Let A and B be the binary random variables attached to their mastery by a learner. We assume the existence of the prerequisite relationship $\mathfrak{A} \to \mathfrak{B}$. The probabilistic*

prerequisite relationship $\mathfrak{A} \rightarrow \mathfrak{B}$ *can be defined with the two following inequalities:*

$$P(B = \neg b \mid A = \neg a) > 0 \qquad P(A = +a \mid B = +b) > 0$$

We remark that the deterministic prerequisite relationship is a particular case of probabilistic prerequisite relationship where the mentioned probabilities are $P(B = \neg b \mid A = \neg a) = 1$ and $P(A = +a \mid B = +b) = 1$. The probabilistic description of the prerequisite relationships gives a first glimpse of the Conditional Probability Distributions (CPDs) ruling the E-PRISM learner model.

The main purpose of the learner model in the E-PRISM framework is to diagnose the learner's knowledge state, which can be derived from inference. We detail the mathematical formulation of the E-PRISM learner model in Section 4.2, and we particularly focus on the tasks of inference and parameter learning in Chapter 5. Overall, E-PRISM is designed to provide the domain and learner models for ITS but is not concerned with teaching and communication models. E-PRISM can be integrated with existing teaching and communication models, or new ones can be developed to work with E-PRISM.

4.1.2 A multilayered organization of E-PRISM components

The structure of the E-PRISM framework is, in practice, more complex than a decomposition into the domain and the learner models. Indeed, the information contained in tutoring systems goes beyond the elements described until now. E-PRISM relies on a multilayer organization of its elements. In particular, we differentiate the resource, domain, and learner layers. Figure 4.1 is the UML diagram representing the E-PRISM framework.

The E-PRISM framework has been developed as an open-source `Python` package, freely available on GitHub[1]. This diagram provides a detailed overview of the main functionalities of the package, which provides tools and resources for building student models in ITS. E-PRISM is a `Python` package describing the elements of usual learner models in ITS with pythonic objects. It offers a flexible and robust framework for building student models in ITSs, and `Python` makes it easy to use and extend for researchers and developers alike.

Resource layer

First, the resource layer relates to the content of the ITS onto which E-PRISM would be applied. This content is composed of static or interactive learning resources on which students learn the domain concepts or practice their skills. For instance, the interactive resources regroup the exercises a student can realize on the ITS. The ITS to which E-PRISM is applied is supposed to rely on interactive exercises: a student can answer questions of the

[1] https://github.com/olivierallegre/e-prism

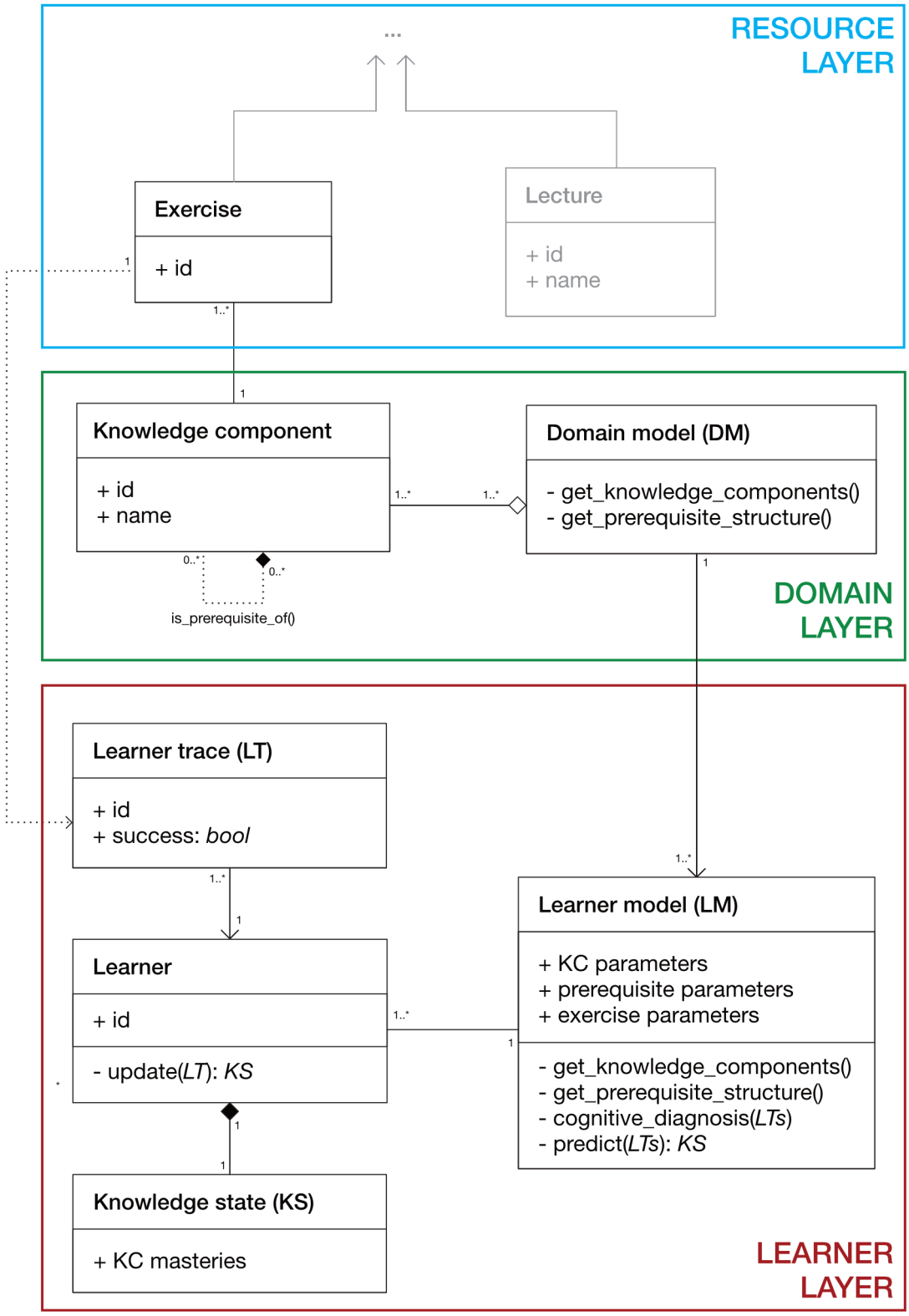


Figure 4.1: UML diagram of the E-PRISM framework architecture. The framework is decomposed into three layers that interact with each other: the resource layer, the domain layer, and the learner layer.

exercise, and she gets feedback from her answer. Nevertheless, these interactive resources can be associated with static resources that aim to provide students with the required knowledge, such as classic lectures or videos on the corresponding subjects.

The E-PRISM framework stores the characteristics related to each element of the resource layer. It may also contain every hierarchical link between its elements, relating to the organization of resources in the educational context. For instance, in an ITS designed to follow the instructions from the Department of Education, the hierarchical structure would describe the advised learning program. While the resource layer in this work will only be composed of exercise objects, we can assume it can be augmented by relationships one can encounter in frameworks like Knowledge Graphs [HBC+21] in the future. For example, the hyperlinks between the resources or the similitudes between the content of the resources could be considered. Overall, the resource layer contains the additional information induced by the environment of the ITS in which E-PRISM would be applied.

In this work, we use a minimal resource layer as we want our results to be as general as possible and potentially applied to numerous ITS. The resource layer is then uniquely composed of exercises on which learner traces are collected.

Domain layer

The domain layer embraces the domain model. As reported in Section 4.1.1, the E-PRISM framework assumes the decomposition of domain knowledge into knowledge components. Each KC is represented in the domain layer with its proper characteristics, which are, in our context, IDs and names.

The KCs in the domain layer are related to exercises in the resource layer. We assume that every question is related to a unique knowledge component. Our work relies on this major hypothesis, which becomes a constraint for the ITS to which E-PRISM is applied.

The prerequisite relationships between KCs are also stored in the domain layer. They are represented symbolically, as the domain model is supposed to represent the ideal knowledge.

Learner layer

The learner layer encompasses the learner model attached to every student. Students are the users of the ITS to which the E-PRISM framework is applied. The learner layer then regroups all the information on them. We associate learners in groups and assume the learners of the same group share the same learner model to simplify the students' modeling. This approach can be interpreted as a stereotypical approach to the learner model, a sub-type of overlay model discussed in Section 2.1.1. Each learner is associated with a unique learner model, depending on the learner's characteristics. Her knowledge state is also stored in the learner layer. It represents the mastery of every KCs in the domain layer with a probability vector.

Learner models do not only contain the parameters associated with the attached learners but also those associated with each other layers' element depending on learners, such as learner-individualized exercise parameters [YKG13] or learner-individualized KC parameters [PH11]. The learner model is attached to two private methods. These methods are the machinery allowing to compute predictions in E-PRISM. They correspond to the machine learning techniques employed to perform the update and prediction equations depicted in Figure 2.1. They aim to answer two queries in the framework. First, determining the learners' knowledge states over time from their traces on the ITS with the `cognitive_diagnosis` method. Second, predicting the learners' future performance with the `predict` method.

In addition, the learner layer contains the learner traces collected on the ITS. Each trace relates to a learner and an exercise in the resource layer. Using the predictive methods of the learner model with the traces of a particular learner allows updating her knowledge state from her learner traces with the `update` method. The E-PRISM framework can process the data practically to perform computations.

4.2 Detailing the implementation of the E-PRISM learner model

Student modeling is essential for providing ITS with personalized feedback functionalities. Designing effective instructional interventions requires diagnosing learners' current knowledge states and predicting their future performance. In E-PRISM, the learner model is responsible for updating learners' knowledge states based on new data, such as their interactions with the system or performance on assessments.

This section provides an in-depth look at the learner model in E-PRISM. First, we detail the modeling of the causal effect of the KC prerequisite structure on the learners' knowledge state at a fixed time. We discuss the use of ICI-model Conditional Probability Distributions (CPDs) to represent the relationships between KCs. These CPDs help modeling the prerequisite structure of domain knowledge through highly interpretable parameters. We compare our model with usual relationship modeling approaches to elaborate on its main benefits. Second, we explain how these CPDs are integrated over time using Dynamic Bayesian Networks (DBNs). DBNs enable the modeling of student knowledge's evolution over time and provide a comprehensive representation of student learning. Finally, we discuss the benefits and downsides of the approach of the E-PRISM framework for learner modeling.

4.2.1 Knowledge modeling at fixed time

First, we focus on modeling the causal effect of prerequisite relationships in the E-PRISM framework, assuming a fixed point in time. In the first place, we assume time is invariant: we will not consider the temporal dimension of the knowledge state.

Under this assumption, we can represent knowledge states using a set of N_{KC} binary random variables, where N_{KC} is the number of KCs in the domain model. Each KC $\mathfrak{X}$ has a corresponding variable, X, representing the learner's mastery of $\mathfrak{X}$. This approach allows us to model the mastery of each KC separately and examine the relationship between them at a fixed point in time.

Naive Bayesian network approaches for prerequisite relationship modeling

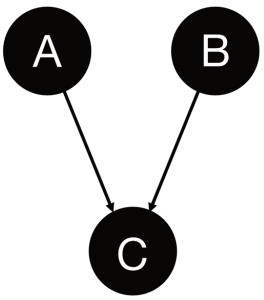
We have seen in Chapter 3 that Bayesian Networks (BNs) are PGMs commonly used to model complex systems. BNs rely on the probability theory and represent the set of dependencies between the modeled variables with a directed graphical structure. Therefore, they are widely used to represent systems exhibiting causal probabilistic relationships. The main advantage of BNs is that they offer great readability for users. Moreover, BN parameters have considerable interpretability as they are directly related to a tangible conditional probability [MBL21].

The naive approach for modeling prerequisite relationships using a Bayesian network would

be constructing a Direct Acyclic Graph (DAG) composed of a vertex for the mastery of each KC and an edge for each prerequisite relationship between two KCs. Definition 4.1.2 defines prerequisite relationships regarding two conditional probabilities. For a given KC $\mathfrak{X}$, the random variable X is either dependent on the random variables that represent the mastery of its parents $\mathbf{Pa}_{\mathfrak{X}} = (Pa_{\mathfrak{X},i})_{i\in[\![1,N_{\text{parents}}(\mathfrak{X})]\!]}$, with probability distribution $P(X|\mathbf{Pa}_{\mathfrak{X}})$, or on the random variables representing the mastery of its children $\mathbf{Ch}_{\mathfrak{X}} = (Ch_{\mathfrak{X},i})_{i\in[\![1,N_{\text{children}}(\mathfrak{X})]\!]}$, with probability distribution $P(X|\mathbf{Ch}_{\mathfrak{X}})$. In the BN supposed to represent the prerequisite relationships between KCs, the choice between these two points of view expresses the chosen causality represented by the network structure. The representation either entails the "parents to children" or the "children to parents" causality.

This work assumes the "parents to children" causality, corresponding to the common approach for prerequisite relationships. The naive Bayesian network approach for modeling prerequisite relationships with the "parents to children" causality would have $2^{N_{\text{parents}}(\mathfrak{X})}$ parameters for each KC.

Example 4.2.1 (Naive BN with the "parents to children" causality)**.** *Let be three knowledge components $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ such that there are two prerequisite relationships $\mathfrak{A} \to \mathfrak{C}$ and $\mathfrak{B} \to \mathfrak{C}$. We introduce a naive Bayesian network that models A, B, and C (which are respectively the random variables encoding the mastery of $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$). The BN represents the effect of their prerequisite relationships with arcs from parents to children. In Figure 4.2, four parameters are needed to model the prerequisite relationships between KCs. These parameters are the conditional probabilities of C given the possible configurations of A and B.*



A	B	P(C|A, B)
0	0	α_1
0	1	α_2
1	0	α_3
1	1	α_4

Figure 4.2: Naive Bayesian network that would represent the mastery of KCs $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ with prerequisite relationships $\mathfrak{A} \to \mathfrak{C}$ and $\mathfrak{B} \to \mathfrak{C}$ in the causality "parents to children". The table attached to it is the corresponding CPD that fully describes the variable C.

Still, we notice the possibility of rather considering the "children to parents" causality for the prerequisite structure. The BN would then have $2^{N_{\text{children}}(\mathfrak{X})}$ parameters for each KC.

Example 4.2.2 (Naive BN with the "children to parents" causality)**.** *We suppose the domain model is the same as in Example 4.2.1. The naive Bayesian network that models the mastery of $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ and the effect of their prerequisite relationships with the "children to parents" causality is represented in Figure 4.3. The parameters associated with the prerequisite structure are the conditional probabilities of A and B given the state of C.*

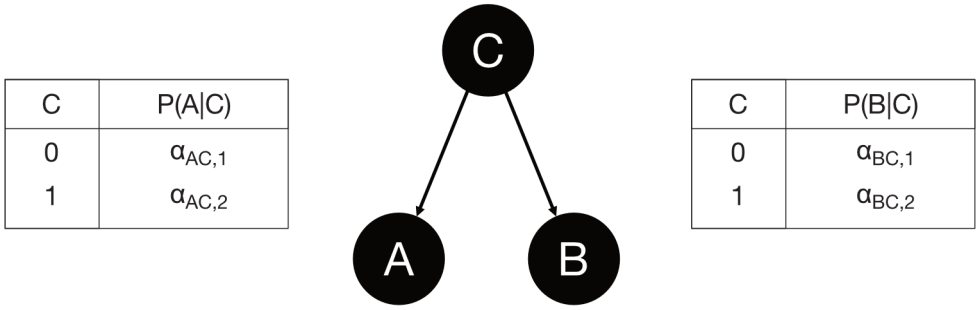


Figure 4.3: Naive Bayesian network that would represent the mastery of KCs $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ with prerequisite relationships $\mathfrak{A} \to \mathfrak{C}$ and $\mathfrak{B} \to \mathfrak{C}$ in the "children to parents" causality. The tables attached to it are the CPDs that fully describe the variables A and B.

Clause of independence of causal influence to reduce the number of parameters

Whatever the chosen causality, naive Bayesian network modeling leads to a large number of parameters when applied to a domain model containing various prerequisite relationships. The number of parameters in the BN exponentially grows with the number of parents. This could induce difficulties in performing tractable inference about the state of the model variables. From a computational perspective, using classic tabular CPDs without constraints could cause troubles when updating the learner model.

We saw in Chapter 3 that the CPDs of a Bayesian network could be constrained to reduce the number of parameters and, consequently, enhance the tractability of the inference and parameter learning tasks. We hypothesize that the model follows the clause of Independence of Causal Influences (ICI) for modeling prerequisite relationships. ICI models aim at reducing the number of parameters needed to specify CPDs [HB96] – see Section 3.1.2.

We recall the main principles of ICI-model CPDs by comparing them to the naive Bayesian network approach for modeling prerequisite relationships. Consider a naive Bayesian classifier between latent random variables $(X_i)_{i\in[\![1,N]\!]}$ and an observable variable Y with a standard tabular CPD. This BN is represented on the left in Figure 4.4, and relies on 2^N parameters $(\alpha_i)_{i\in[\![1,2^N]\!]}$.

On the other hand, ICI models integrate an auxiliary variable for each "cause" variable and define the "consequence" variable as a deterministic function of these auxiliary variables. This approach assumes that there are no interactions among the causal mechanisms by which the "cause" variables affect the value of the "consequence" variable. The auxiliary variable introduced by an ICI model between a cause variable X_i and a consequence variable Y is usually denoted $Z_{Y,i}$. The CPD of each random variable is defined as a deterministic function of the causal influences from its parents through variables $(Z_{Y,i})_i$. The ICI-model CPD is elicited in Definition 3.1.15. The Bayesian network with ICI-model CPDs modeling

the naive Bayesian network dependencies is represented on the right in Figure 4.4. It relies on $2N$ parameters $(s_i, q_i)_{i \in [\![1,N]\!]}$.

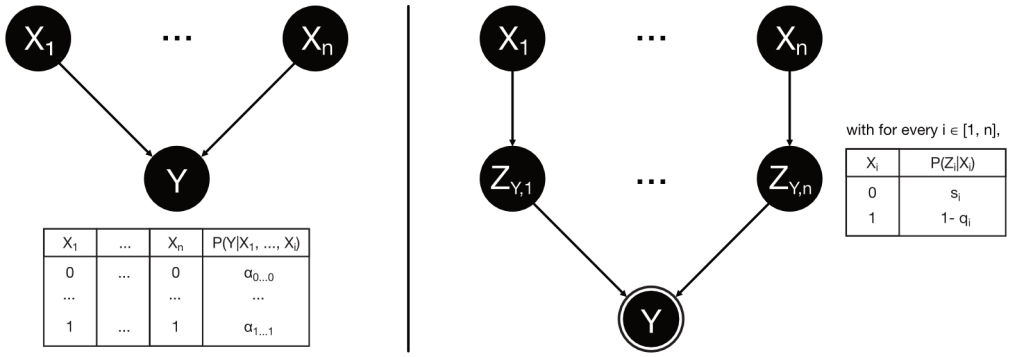


Figure 4.4: Comparison between the naive Bayesian network classifier representing the causal influence of the variables $(X_i)_i$ on Y (on the left), and the equivalent Bayesian network based on an ICI model (on the right). The double line represents that Y is defined as a deterministic function of the $(Z_{Y,i})_i$.

When modeling prerequisite relationships with ICI-model CPDs, the number of parameters in the ICI-model CPD will depend on the chosen causality. For the "parents to children" causality, the ICI-model CPD of a variable X associated with the KC $\mathfrak{X}$ would have $2N_{\text{parents}}(\mathfrak{X})$ parameters. Reciprocally, for the "children to parents" causality, the ICI-model CPD of a variable X associated with the KC $\mathfrak{X}$ would have $2N_{\text{children}}(\mathfrak{X})$ parameters. The number of parameters of an ICI-model CPD is at most linear with the maximum number of parents in the domain model. This highlights the interest in ICI-based Bayesian networks in terms of interpretability.

Notation 4.2.1 (ICI-based Bayesian network)**.** *We call "ICI-based Bayesian network" every BN for which an ICI model CPD models every variable.*

4.2.2 Motivations for choosing the causality

The choice of causality when instantiating the ICI-based Bayesian networks modeling the prerequisite relationships not only influences the number of parameters in the learner model but also determines which deterministic function to use in the ICI-model CPD. The CPDs in ICI models are indeed applications of a deterministic function to the auxiliary variables representing the causal influences of parents. We have presented the main deterministic functions used in ICI models in Table 3.1. The function to use depends on the causality chosen to represent the prerequisite relationships between KCs.

We first present the Noisy-AND gate, an ICI-model CPD with an AND deterministic function that models prerequisite relationships with the "parents to children" causality. Then, we present the Noisy-OR gate, an ICI-model CPD with an OR deterministic function that models prerequisite relationships with the "children to parents" causality. Finally, we explain why we have opted for Noisy-AND gates.

Noisy-AND gate

Let $\mathfrak{X}$ be a KC attached with the random variable X representing the learner's mastery of $\mathfrak{X}$. The deterministic definition of a prerequisite relationship states that for each parent $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$ of $\mathfrak{X}$, the non-mastery of $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$ implies the non-mastery of $\mathfrak{X}$. We have $\forall i$, $(Pa_{\mathfrak{X},i} = \neg pa_i) \Rightarrow (X = \neg x)$. The probabilistic prerequisite relationship definition adds there exists a positive probability that this implication is not effective. This can be interpreted as a deterministic AND function applied to the causal influence of each parent. As a result, if any parent of $\mathfrak{X}$ is not mastered, it is highly probable $\mathfrak{X}$ will not be mastered as well. The mastery of $\mathfrak{X}$ would be modeled as an AND function of the auxiliary variables representing the causal influence of each parent $Pa_{\mathfrak{X},i}$. We call this model Noisy-AND gate.

We define the Noisy-AND gate of $\mathfrak{X}$ as the Bayesian network $\mathcal{B}_{\text{Noisy-AND}}(\mathfrak{X})$, represented in Figure 4.5. $\mathcal{B}_{\text{Noisy-AND}}(\mathfrak{X})$ considers variables for the mastery of $\mathfrak{X}$ and of each of its parents $(\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}})_i$. It also introduces auxiliary variables which represent the noise on the causal effect of the mastery of each parent on the mastery of $\mathfrak{X}$. For $i \in [1, N_{\text{parents}}(\mathfrak{X})]$, $Z_{\mathfrak{X},i}$ is the variable associated to the event "There exists a causal effect of $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$'s mastery on $\mathfrak{X}$'s mastery." In $\mathcal{B}_{\text{Noisy-AND}}(\mathfrak{X})$, $Z_{\mathfrak{X},i}$ is the target of an arc from $Pa_{\mathfrak{X},i}$ and the source of an arc to X.

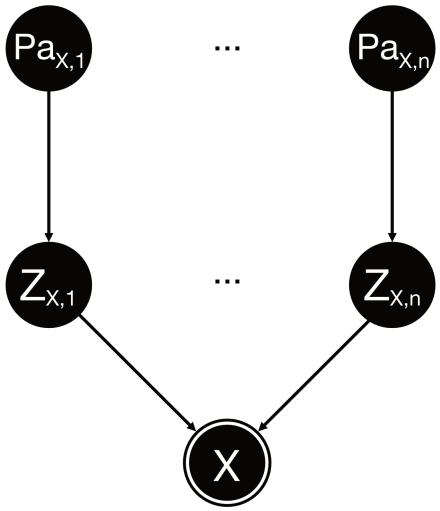


Figure 4.5: DAG structure of the Noisy-AND gate of $\mathfrak{X}$ with prerequisite KCs $(\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}})_i$. CPDs that rule the DAG structure are given in Tables 4.1a and 4.1b.

The causal effect represented by $Z_{\mathfrak{X},i}$ can be inhibited or substituted [DD06]. It is inhibited if the "cause" on $Pa_{\mathfrak{X},i}$ no longer induces the consequence on X. We denote $q_{\mathfrak{X},i}$ the probability that there is an inhibitor of the causal effect of $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$'s mastery on $\mathfrak{X}$'s mastery. This causal effect is substituted if the "cause" $Pa_{\mathfrak{X},i}$ is no longer required for the "consequence" X. We denote s_i the probability that there exists a substitute to the causal effect of $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$'s mastery on $\mathfrak{X}$'s mastery. The CPD of $Z_{\mathfrak{X},i}$ is represented in Table 4.1a. We discuss the interpretability of these parameters in Section 2.2.4. In the Noisy-AND gate of $\mathfrak{X}$, the CPD of X is an AND

function of all variables $(Z_{\mathfrak{X},i})_{i\in[\![1,N_{\text{parents}}(\mathfrak{X})]\!]}$, represented in Table 4.1b.

$Pa_{\mathfrak{X},i}$	$P(Z_{\mathfrak{X},i} = \neg z_i \mid Pa_{\mathfrak{X},i})$	$P(Z_{\mathfrak{X},i} = +z_i \mid Pa_{\mathfrak{X},i})$
$\neg pa_i$	$1 - s_{\mathfrak{X},i}$	$s_{\mathfrak{X},i}$
$+pa_i$	$q_{\mathfrak{X},i}$	$1 - q_{\mathfrak{X},i}$

(a) CPD of the auxiliary node $Z_{\mathfrak{X},i}$

$\forall i, Z_{\mathfrak{X},i} = +z_i$	$P(X = \neg x \mid \{Z_{\mathfrak{X},i}\}_i)$	$P(X = +x \mid \{Z_{\mathfrak{X},i}\}_i)$
0	1	0
1	0	1

(b) CPD of X

Table 4.1: CPDs in the Noisy-AND gate of $\mathfrak{X}$.

Noisy-OR gate

On the other hand, Definition 4.1.1 asserts that the mastery of each child $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$ of $\mathfrak{X}$ implies the mastery of $\mathfrak{X}$. We have $\forall i$, $(Ch_{\mathfrak{X},i} = +ch_i) \Rightarrow (X = +x)$. Definition 4.1.2 introduces a positive probability that the implication is ineffective. The CPD of a variable X, representing the learner's mastery of $\mathfrak{X}$, can be interpreted as an OR function applied to the causal influence of each child $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$. If any child of $\mathfrak{X}$ is mastered, it is highly probable that $\mathfrak{X}$ will be mastered as well. The CPD of X is a Noisy-OR gate.

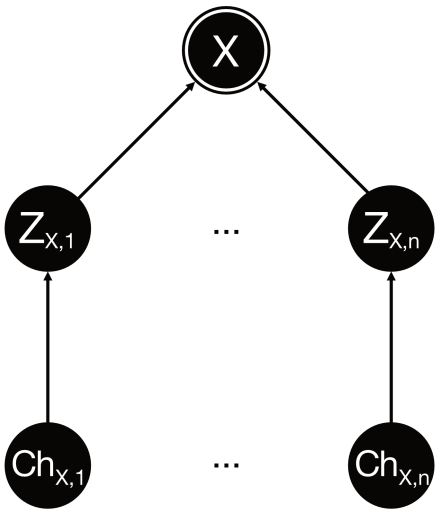


Figure 4.6: DAG structure of the Noisy-OR gate of the knowledge component $\mathfrak{X}$ with prerequisite KCs $(\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}})_i$. CPDs that rule the DAG structure are given in Tables 4.2a and 4.2b.

We define the Noisy-OR gate of $\mathfrak{X}$ as the Bayesian network $\mathcal{B}_{\text{Noisy-OR}}(\mathfrak{X})$, represented in

Figure 4.6. $\mathcal{B}_{\text{Noisy-OR}}(\mathfrak{X})$ considers variables for the mastery of $\mathfrak{X}$ and of each of its children $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$. It integrates auxiliary variables representing the causal effect of the mastery of each child on the mastery of $\mathfrak{X}$. For $i \in [1, N_{\text{children}}(\mathfrak{X})]$, $Z_{\mathfrak{X},i}$ is a variable associated to the event "There exists a causal effect of $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$'s mastery on $\mathfrak{X}$'s mastery." In $\mathcal{B}_{\text{Noisy-OR}}(\mathfrak{X})$, $Z_{\mathfrak{X},i}$ is the target of an arc from $Ch_{\mathfrak{X},i}$ and the source of an arc to X.

In Noisy-OR gates, the causal effect represented by $Z_{\mathfrak{X},i}$ is not substituted [DD06]. We denote $q_{\mathfrak{X},i}$ the probability that there is an inhibitor of the causal effect of $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$ mastery on $\mathfrak{X}$ mastery. The CPD of $Z_{\mathfrak{X},i}$ is represented in Table 4.2a. The CPD of X is an OR function of all variables $(Z_{\mathfrak{X},i})_{i\in[\![1,n_{\text{children}}(\mathfrak{X})]\!]}$, represented in Table 4.2b.

$Ch_{\mathfrak{X},i}$	$P(Z_{\mathfrak{X},i} = \neg z_i \mid Ch_{\mathfrak{X},i})$	$P(Z_{\mathfrak{X},i} = +z_i \mid Ch_{\mathfrak{X},i})$
$\neg ch_i$	1	0
$+ch_i$	$q_{\mathfrak{X},i}$	$1 - q_{\mathfrak{X},i}$

(a) CPD of the auxiliary node $Z_{\mathfrak{X},i}$

$\exists i, Z_{\mathfrak{X},i} = +z_i$	$P(X = \neg x \mid \{Z_{\mathfrak{X},i}\}_i)$	$P(X = +x \mid \{Z_{\mathfrak{X},i}\}_i)$
0	1	0
1	0	1

(b) CPD of X

Table 4.2: CPDs in the Noisy-OR gate of $\mathfrak{X}$.

Choosing the deterministic function

As stated in Section 4.2.1, we have chosen to consider the "parents to children" causality rather than the "children to parents" causality because it sounds like the most intuitive causality.

Nevertheless, the Noisy-AND and Noisy-OR gate CPDs are quite similar. Consequently, each technique detailed in the following should apply to the Noisy-OR representation even if we have decided to drop Noisy-OR gates from this study. The Noisy-OR representation is notably implemented in the `Python` package of the E-PRISM framework.

Notation 4.2.2 (Auxiliary variables in the Noisy-AND gate)**.** *In the following, we denote $Z_{\mathfrak{X}_{target},\mathfrak{X}_{source}}$ the auxiliary variable between the variables representing the masteries of two KCs $\mathfrak{X}_{source}$ and $\mathfrak{X}_{target}$. The auxiliary variable between the masteries of $\mathfrak{X}$ and its i-th parent is denoted $Z_{\mathfrak{X},i}$. The auxiliary variable between the masteries of $\mathfrak{X}$ and its i-th children is denoted $Z_{i,\mathfrak{X}}$.*

4.2.3 Modeling the knowledge state dynamics

So far, we have modeled the learner's knowledge state at a fixed time, including the causal effect of the prerequisite structure. However, it is necessary to incorporate the time dimension in the modeling of KC masteries to capture the dynamics of the learning process. We have seen in Chapter 2 that many learner models already take this approach, such as Bayesian Knowledge Tracing (BKT) and derivatives [CA94, PH10, PH11, YKG13, KKSG17], or logistic regression models [CPBV19]. To accurately capture this aspect in our model, we assume the KC mastery variables to be time-dependent.

Notation 4.2.3 (Temporal random variable for the mastery of a knowledge component)**.** *Let be $\mathfrak{X}$ a knowledge component. We denote X^t, the binary random variable associated with the mastery of $\mathfrak{X}$ at the timeslice t.*

In this section, we discuss how we model the continuity of the mastery of a particular KC. We wonder how the E-PRISM learner model incorporates the causal effect of the domain model prerequisite structure into learners' knowledge states over time. In particular, we will recall the implications of using Dynamic Bayesian Networks (DBNs) and detail how such a technique can be used in our context.

Using a dynamic Bayesian network

Building upon the previous discussion on the dynamics of the learning process, we turn to the use of Dynamic Bayesian Networks (DBNs) to model learners' knowledge states over time. DBNs are a powerful tool for modeling dynamic systems as they allow us to capture temporal dependencies. It particularly fits for modeling the relationship between the knowledge state of a learner at different points in time.

In this work, we use a stationary DBN. Thus, we assume that the learners' knowledge states follow the Markov assumption, meaning that the knowledge state at time t only depends on itself and the knowledge state at time $t-1$ – see Definition 3.1.20. Moreover, we also assume the system is stationary, meaning that the conditional probability distribution describing the transition between the knowledge state at time $t-1$ and time t is time-independent.

We have defined in Definition 3.1.22 that a stationary DBN can be fully described by a set of two Bayesian networks $\langle \mathcal{B}_0, \mathcal{B}_{\rightarrow} \rangle$. $\mathcal{B}_0$ represents the prior knowledge state at time $t = 0$, while $\mathcal{B}_{\rightarrow}$ models the temporal dependencies between the knowledge state at consecutive points in time. The following details the implementation of $\mathcal{B}_0$ and $\mathcal{B}_{\rightarrow}$ composing the learner model.

Initial bayesian network of the learner model

The initial Bayesian network $\mathcal{B}_0$ represents the initial knowledge state of the learner. It comprises a Noisy-AND gate for each knowledge component in the domain model. These Noisy-AND gates share common KC mastery variables since prerequisite relationships exist

between the KCs. The structure of the Noisy-AND gates forms a Bayesian network, depicted in Figure 4.7. Specifically, the network is composed of a variable for each knowledge component, as well as the auxiliary variables $(Z^t_{\mathfrak{X},i})_{i\in[\![1,N_{\text{parents}}(\mathfrak{X})]\!]}$ for every KC $\mathfrak{X}$.

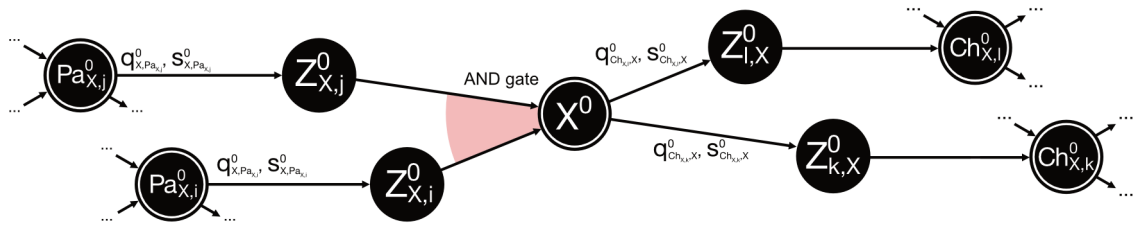


Figure 4.7: DAG structure of the initial Bayesian network $\mathcal{B}_0$ centered on the variable X^0 representing the mastery of a KC $\mathfrak{X}$ at time $t = 0$.

Transition bayesian network of the learner model

The transition Bayesian network $\mathcal{B}_\rightarrow$ represents the transition of the learner's knowledge state between two successive timeslices. It reports how the knowledge state at time $t-1$ influences the knowledge state at time t. Similarly to $\mathcal{B}_0$, we assume that the knowledge state at time t is represented by a set of interrelated Noisy-AND gates for each knowledge component of the domain model. Thus, $\mathcal{B}_\rightarrow$ integrates the auxiliary variables $(Z^t_{\mathfrak{X},i})_{i\in[\![1,N_{\text{parents}}(\mathfrak{X})]\!]}$ for each KC $\mathfrak{X}$.

$\mathcal{B}_\rightarrow$ also considers a variable for each knowledge component of the domain model at time $t-1$. The variable X^{t-1} is integrated to the Noisy-AND gate of X^t in the same way as the variables $Pa^t_{\mathfrak{X},i}$ for $i \in [1, N_{\text{parents}}(\mathfrak{X})]$, as a stakeholder that leads to X^t. We introduce an auxiliary variable $T^t_{\mathfrak{X}}$ between X^{t-1} and X^t. $T^t_{\mathfrak{X}}$ represents causal effect between the mastery of a knowledge component $\mathfrak{X}$ on two successive timesteps $(t-1, t)$. It is associated with the event "There is a causal effect of the mastery of $\mathfrak{X}$ at timestep $t-1$ on the mastery of $\mathfrak{X}$ at timestep t." We assume that the causal effect of the mastery of $\mathfrak{X}$ between two successive timesteps can be inhibited or substituted. We denote $f_{\mathfrak{X}}$, the probability there exists an inhibitor of this causal effect, and $l_{\mathfrak{X}}$, the probability there exists a substitute for this causal effect. The CPD of $T^t_{\mathfrak{X}}$ is represented in Table 4.3.

X^{t-1}	$P(T^t = \neg t \mid X^{t-1})$	$P(T^t = +t \mid X^{t-1})$
$\neg x$	$1 - l_{\mathfrak{X}}$	$l_{\mathfrak{X}}$
$+x$	$f_{\mathfrak{X}}$	$1 - f_{\mathfrak{X}}$

Table 4.3: Conditional probability table of the auxiliary node $T^t_{\mathfrak{X}}$ that representing the causal effect of X^{t-1} on X^t.

The CPD of X^t is an AND function on variables $T^t_{\mathfrak{X}}$ and $\{Z^t_{\mathfrak{X},i} |\ i \in [\![1, N_{\text{parents}}(\mathfrak{X})]\!]\}$. We

represent $\mathcal{B}_{\rightarrow}$ with a focus on a KC $\mathfrak{X}$ and its parents and children in Figure 4.8.

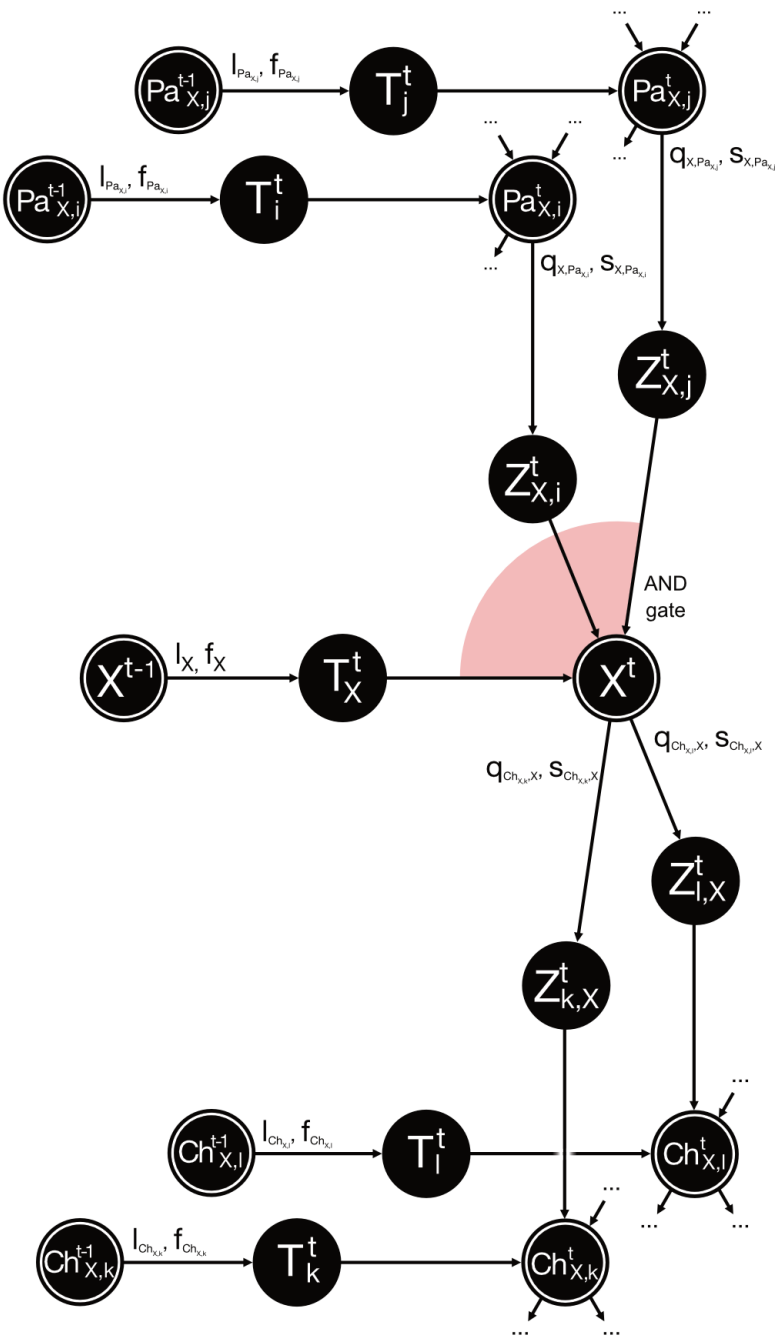


Figure 4.8: DAG structure of the transition Bayesian network $\mathcal{B}_{\rightarrow}$ centered on the variable X^t representing the mastery of a KC $\mathfrak{X}$ at any time $0 < t < T$.

The transition Bayesian network $\mathcal{B}_{\rightarrow}$ captures the dynamic nature of the learning process by considering that the mastery of a knowledge component X at timestep t not only depends on the mastery of X's parents but also on the mastery of X at previous timestep $t-1$.

Unrolled representation of the learner model

The DBN of the E-PRISM learner model is fully depicted from the definition of the initial and transition Bayesian networks. It is a compact description of the DBN.

Example 4.2.3 (Initial Bayesian network and transition Bayesian network)**.** *We get back to Example 4.2.1, which studies the mastery of the KCs* $\mathfrak{A}$, $\mathfrak{B}$, *and* $\mathfrak{C}$. *We represent the initial and transition Bayesian networks describing the dynamic Bayesian network of the E-PRISM learner model corresponding to this domain model in Figure 4.9.*

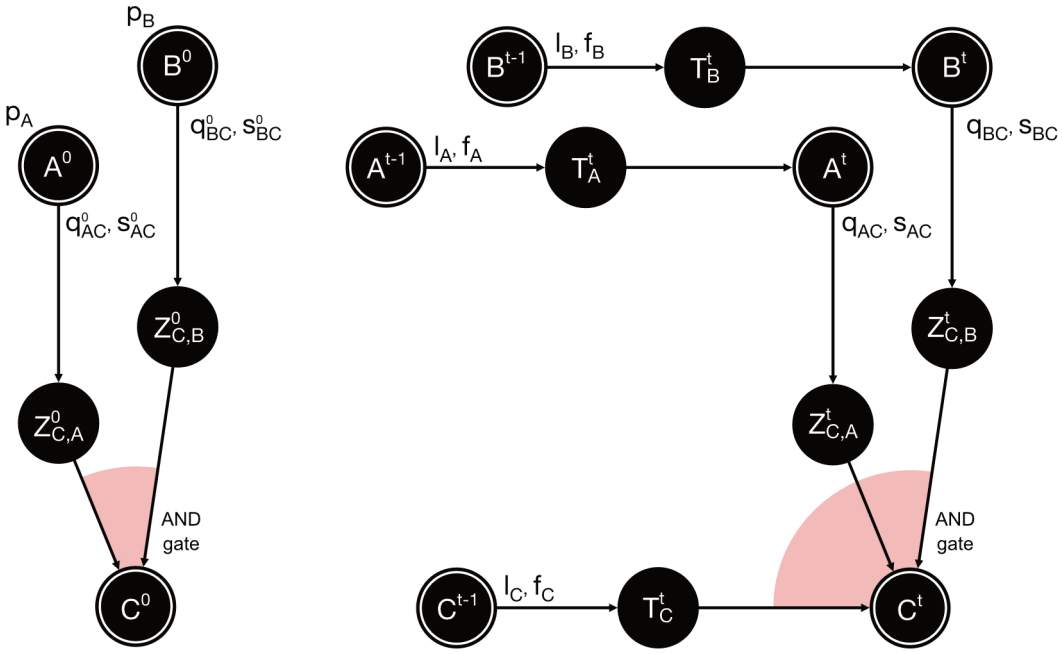


Figure 4.9: DAG structure of the initial Bayesian network $\mathcal{B}_0$ and the transition Bayesian network $\mathcal{B}_{\rightarrow}$ representing the learner model used to infer knowledge states on the domain model described in Example 4.2.1 in E-PRISM.

DBNs may have a compact representation, but they can still be considered a large network, as they can be unrolled to any length. A DBN is essentially the combination of the initial Bayesian network $\mathcal{B}_0$ and a potentially unlimited number of instances of the transition Bayesian network $\mathcal{B}_{\rightarrow}$.

Example 4.2.4 (Unrolled representation of the learner model's DBN)**.** *The dynamic Bayesian network described in Example 4.2.3 can be unrolled when considering a given number of transactions from the learner. We represent the unrolled dynamic Bayesian network of the learner model for three timesteps in Figure 4.10.*

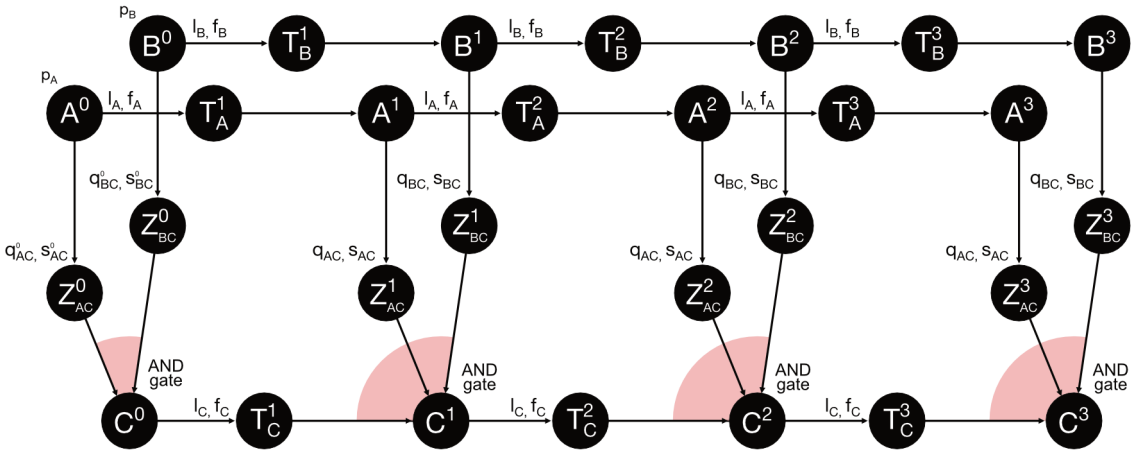


Figure 4.10: DAG structure of the unrolled dynamic Bayesian network representing the learner model used to infer knowledge states on the domain model described in Example 4.2.1 in E-PRISM. Three timeslices are considered in the example.

4.2.4 Benefits and downsides of the learner model

The learner model in the E-PRISM framework works with dynamic Bayesian networks. It has several benefits, particularly concerning its parameters. First, we show that using Noisy-AND gates to represent the evolution of the learner's knowledge state allows a significant reduction of the number of parameters. Additionally, we highlight that the parameters in the DBN improve the model interpretability compared to other machine learning models.

Number of parameters and tractability of the E-PRISM learner model

On the one hand, using ICI-model CPDs drastically reduces the number of model parameters compared to the classic Bayesian network formulation [ZVD06]. Large Bayesian networks, specifically the dynamic ones like in Käser et al.'s work [KKSG17], potentially imply many parameters. We showed in Section 4.2.1 that the number of parameters of the CPD attached with a given knowledge component $\mathfrak{X}$ changes from exponential dependence with the number of $\mathfrak{X}$'s parents with the classic BN approach to linear dependence with the ICI-model CPDs approach. Therefore, there is a significant decrease in the number of parameters in the DBN.

For instance, Käser et al. also introduce a dynamic Bayesian network to trace students' knowledge over time. Käser et al. employ classic tabular CPDs in the network, where there are $2^{N_{\text{parents}}(\mathfrak{X})+1}$ parameters for each KC $\mathfrak{X}$ in the transition BN of the DBN [KKSG17]. On the contrary, the learner model in E-PRISM indeed only relies on $2(N_{\text{parents}}(\mathfrak{X})+1)$ parameters for each KC $\mathfrak{X}$ in the transition BN thanks to the auxiliary variables. Therefore, the gap in the number of parameters increases with the number of KCs considered in the model. The difference may skyrocket when multiple prerequisite relationships are considered.

An enhanced interpretability of parameters

Compared to other machine learning models, the parameters of Bayesian networks are already highly interpretable. They represent conditional probabilities on potentially well-designed variables. The parameters of the ICI-model CPDs in the E-PRISM learner model are even more insightful. Each parameter is only related to a pair of variables due to the clause of independence of causal influences.

In our model, $l_{\mathfrak{X}}$ and $f_{\mathfrak{X}}$ are the parameters associated with the pair of variables (X^{t-1}, X^t) for $t > 0$. $q_{\mathfrak{X},i}$ and $s_{\mathfrak{X},i}$ are the parameters associated with the pair of variables $(Pa^t_{\mathfrak{X},i}, X^t)$ for $t > 0$. The model parameters depict both the causal effect of the temporality of the learning process and the causal effect of the prerequisite relationships between knowledge components. We can note that there is no formal difference between the mathematical formulation of variables $T^t_{\mathfrak{X}}$ and $Z^t_{\mathfrak{X},i}$, as well as between $l_{\mathfrak{X}}$ and $q_{\mathfrak{X},i}$s, and $f_{\mathfrak{X}}$ and $s_{\mathfrak{X},i}$. We differentiate them in the vocabulary to provide better readability to the model.

Indeed, these parameters can be interpreted through real phenomena associated with the context of a KC prerequisite structure during the learning process. On the one hand, $l_{\mathfrak{X}}$ and $f_{\mathfrak{X}}$ represent the transition of the KC mastery between time $t-1$ and t. Similarly to BKT, $f_{\mathfrak{X}}$ corresponds to the probability of forgetting $\mathfrak{X}$ between two timeslices, and $l_{\mathfrak{X}}$ corresponds to the probability of learning $\mathfrak{X}$ between two timeslices. They provide direct measures of the learners' learning and forgetting processes. On the other hand, $q_{\mathfrak{X},i}$ and $s_{\mathfrak{X},i}$ relate the effectivity of the prerequisite relationship between $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$ and $\mathfrak{X}$. $q_{\mathfrak{X},i}$ is the probability that there exists an inhibitor of the causal effect of $Pa^t_{\mathfrak{X},i}$ on X^t for $t > 0$. It represents the probability that the mastery of the i-th parent of $\mathfrak{X}$ is insufficient to master $\mathfrak{X}$ when all other parents are mastered. $s_{\mathfrak{X},i}$ is the probability that there exists a substitute to the causal effect of $Pa^t_{\mathfrak{X},i}$ on X^t for $t > 0$. It represents the probability that the mastery of the i-th parent of $\mathfrak{X}$ is not required to master $\mathfrak{X}$. $q_{\mathfrak{X},i}$ and $s_{\mathfrak{X},i}$ are sensors of the effectiveness of the prerequisite relationship between $\mathfrak{Pa}_{\mathfrak{X},\mathrm{i}}$ and $\mathfrak{X}$ for learners.

E-PRISM is based on parameters that offer comprehensive insights into the interactions between variables. Overall, the parameters in E-PRISM show enhanced interpretability compared to traditional Bayesian network models. Indeed, the parameters in tabular CPDs are associated with very specific and narrow situations. Thanks to ICI-model CPDs, the parameters in the E-PRISM learner model represent wider events. Specifically, it represents the influence of each prerequisite relationship with a pair of parameters. This can then be more understandable for the framework user.

Latent variables

Nevertheless, the E-PRISM learner model has a major drawback. Most learner models consider that the variables representing the learners' knowledge state are hidden. These variables then relate to observable variables through new parameters. On the contrary, the

E-PRISM framework considers the latent KC mastery variables as the observable variables. Consequently, the E-PRISM framework may miss noise sources. For instance, contrary to Bayesian Knowledge Tracing (BKT), the E-PRISM learner model doesn't consider the slip phenomenon when a learner who has mastered a KC can still miss an exercise related to that KC, nor the guess phenomenon when a learner who has not mastered a KC can still succeed on an exercise related to it.

Moreover, the E-PRISM learner model hinges on new latent variables, notably because of ICI-model CPDs. For each KC $\mathfrak{X}$ of the domain model, the auxiliary variables $T_{\mathfrak{X}}$ and $\forall i, Z_{\mathfrak{X},i}$ cannot be observed in any case. This will complexify both inference and learning tasks in the E-PRISM learner model. We provide more details on the cause of these issues in Chapter 5, where we introduce solutions to avoid them.

4.3 Discussion

In this chapter, we presented Embedding Prerequisite Relationships In Student Modeling (E-PRISM), a knowledge-enhanced machine learning framework that mobilizes symbolic and data-driven techniques for providing Intelligent Tutoring Systems (ITS) with student modeling techniques. The E-PRISM framework has been developed to be compatible with the data of a wide range of ITS. The framework has been implemented as an open-source Python package[2], which provides a set of pythonic objects modeling the main ITS elements, such as knowledge components or learner traces.

In particular, we focused on the E-PRISM learner model. It relates to a domain model consisting of Knowledge Components (KCs) related with each other with prerequisite relationships. The E-PRISM learner model is inspired by overlay learner models that consider the learner's knowledge a subset of the domain knowledge. It defines the learner's knowledge state as a vector of KC mastery probabilities. It also integrates prerequisite relationships into the model thanks to an ICI-based dynamic Bayesian network. It considers the causal effects of the prerequisite structure of domain knowledge in addition to the learning and forgetting phenomena. It relies on highly interpretable parameters related to each individual causal effect.

The E-PRISM learner model is a dedicated tool for predicting learners' performance and diagnosing their knowledge state over time. We wonder how to update this learner model from new data. To do so, inference techniques in complex Bayesian networks must be employed. Because of the scarce structure of the learner model's DBN, performing such computations can rapidly be challenging. Chapter 5 provides a deep focus on the techniques and algorithms for diagnosis, prediction, and model update with the E-PRISM framework.

[2] https://github.com/olivierallegre/e-prism

Chapter 5

Functionalities of the E-PRISM learner model

The E-PRISM framework provides tools for modeling students regarding domain knowledge. This chapter details how the E-PRISM learner model can diagnose and predict students' knowledge state from their interaction with an ITS or their performance on assessments. Inference techniques must be applied to the E-PRISM learner model to determine the knowledge states of students over time or other usual ITS functionalities.

First, we review the application of the inference techniques for Bayesian networks detailed in the literature reported in Chapter 3 to E-PRISM. We examine the convergence of these techniques with the Bayesian network modeling approach used by E-PRISM. Then, we propose a new method for approximate inference derived from Blocking Gibbs Sampling based on a variant of the Gibbs sampling algorithm. This method allows us to perform accurate and converging inference tasks. We explain the theoretical basis of the method by providing examples of its application.

This chapter also sheds light on the parameter learning procedure of the E-PRISM learner model. This work assumes the prerequisite structure is known. Still, the values of the E-PRISM learner model parameters must be estimated to provide accurate predictions. We introduce how our new method for approximate inference can be included in a particle-based parameter learning procedure. Finally, we present how to perform parameter learning in the E-PRISM learner model and highlight the convergence of the procedure with synthetic data. Estimating the E-PRISM learner model parameters is crucial, as it provides insights into the prerequisite structure of domain knowledge thanks to the great interpretability of the parameters.

5.1 Diagnosing the learner's knowledge state over time

The E-PRISM learner model is a crucial component for deriving the knowledge state of learners from evidence. It can provide a comprehensive understanding of the learners' performance and knowledge states over time by integrating information from learner traces. The E-PRISM learner model is a dedicated tool for predicting learner performance and diagnosing their knowledge state. This section will explore the application of inference techniques to the Dynamic Bayesian Network (DBN) of the E-PRISM learner model. Inferring the probability distribution of the DBN variables is essential for providing diagnoses of learners' knowledge states.

First, we examine the queries on which these ITS functionalities rely and how they can be translated mathematically. These queries involve the computation of posterior probabilities of the state of students' KC masteries given observed data. They are essential for accurate student modeling and diagnosis with Bayesian networks. Next, we review the inference techniques for Bayesian networks presented in Chapter 3. We highlight the intractability or convergence issues induced by these techniques. For instance, exact inference is intractable on the ICI-based dynamic Bayesian networks used in E-PRISM. Finally, we will introduce a novel approach for approximate inference in Bayesian networks based on Blocking Gibbs Sampling (BGS). BGS approximate inference can overcome some of the other inference techniques' limitations and provides tractable and converging predictions. This approach has been shown to be particularly effective for ICI-based dynamic Bayesian networks, making it a promising method for the inference tasks required in the E-PRISM framework.

5.1.1 Probability queries in E-PRISM

We recall that inference mainly consists in answering probability queries on the model. A probability query is defined in Definition 3.2.3. They introduce query variables and evidence, which depend on the query. The main functionalities expected in an ITS are knowledge state diagnosis and learner performance prediction. These two functionalities are associated with two types of common queries, often performed on a given set of learner traces in E-PRISM. In practice, in E-PRISM, these two queries are very similar, as there are no differences between hidden mastery variables and observable variables.

Evidence variables

We assume the learner only answers one question at a time. Therefore, the evidence variables of the probability queries performed on the E-PRISM learner model consist of one variable per timeslice.

Example 5.1.1 (Evidence variables)**.** *Let* $\mathfrak{A}$*,* $\mathfrak{B}$ *and* $\mathfrak{C}$ *be knowledge components. We assume that each KC is associated with an exercise. Suppose that a learner has given the following answers to the questions of an ITS: incorrect to a question requiring* $\mathfrak{C}$*, then correct to a*

question requiring $\mathfrak{A}$, and finally incorrect to a question requiring $\mathfrak{B}$. The evidence generated from these learner traces is $\{C^0 = 0, A^1 = 1, B^2 = 0\}$.

Diagnosis query

An ITS should diagnose the learner's knowledge state. The first query aims to determine the evolution of the knowledge state over time based on evidence from learner traces. We denote $\mathbf{x}$ the E-PRISM learner model variables and $\mathbf{e}$ the evidence variables. The diagnosis query corresponds to the computation of the probability $P(\mathbf{x} \setminus \mathbf{e} \mid \mathbf{e})$.

Example 5.1.2 (Diagnosis query)**.** *Suppose the evidence presented in Example 5.1.1. The diagnosis query is given in Equation 5.1.*

$$P(\mathbf{y} \mid \mathbf{e}) = P(\mathbf{X}^0 \setminus \{C^0\}, \mathbf{X}^1 \setminus \{A^1\}, \mathbf{X}^2 \setminus \{B^2\} \mid C^0 = 0, A^1 = 1, B^2 = 0) \tag{5.1}$$

Prediction query

Second, an ITS should be able to predict the learner's future performance. We denote T as the last time the learner produced a transaction with the ITS. The prediction probability query corresponds to the computation of the probability $P(\mathbf{X}^{T+1} \mid \mathbf{e})$, where $\mathbf{X}^{T+1}$ is the learner's knowledge state at timeslice $T + 1$. It aims to predict the knowledge state at the timeslice right after the last one informed by the evidence variables.

Example 5.1.3 (Prediction query)**.** *Suppose the evidence presented in Example 5.1.1. The diagnosis query is given in Equation 5.2.*

$$P(\mathbf{y} \mid \mathbf{e}) = P(\mathbf{X}^3 \mid C^0 = 0, A^1 = 1, B^2 = 0) \tag{5.2}$$

These queries are employed in the E-PRISM framework to provide ITS with basic functionalities and allow more complex tasks such as model learning. For instance, the diagnosis query is used for completing scarce data during E-PRISM parameter learning. This will be further detailed in Section 5.2.

5.1.2 Intractibility of exact inference in E-PRISM

We elicited the probability queries that must be answered for E-PRISM to provide the main ITS functionalities. We wonder now how exact inference can handle them. Performing exact inference for these two types of probability queries in E-PRISM can be challenging as they involve a large state space for the query variables. We detail the cause of such intractability and show how the intractability is observed in practice.

Source of intractability

The first approach to answer these probability queries is to use exact inference techniques in the DBN of E-PRISM. Exact inference techniques in a Bayesian network have been developed in Chapter 3. In particular, we saw exact inference is NP-hard, making it intractable when the set of query variables is large. Computing the joint posterior of query variables indeed consists of computing the probability related to each possible combination of the variable states. The space dimension of the target joint posterior is $2^{N_{\text{query}}}$, with N_{query} the number of query variables.

The exact inference techniques can be sufficient when answering prediction probability queries, such as in Equation 5.2, because the state space of the query variables is small enough. However, when considering the probability queries on the whole knowledge state, such as in Equation 5.1, the size of the set of query variables can become an issue.

The structure of the E-PRISM learner model detailed in Chapter 4 intrinsically implies a high scarcity of the data. The auxiliary variables either represent the causal effect of KCs on each other or the causal effect of the temporal transition of knowledge mastery. They are latent variables of the DBN and cannot be observed. Thus, the number of query variables for querying the learner's knowledge state over time can become enormous when the learner has resolved many exercises. Specifically, with N_{traces} the number of learner traces, N_{KCs} the number of KCs in the domain model, and $N_{\text{prerequisites}}$ the number of prerequisite relationships in the domain model, there are $N_{\text{KCs}} + N_{\text{prerequisites}} - 1 + N_{\text{traces}}(2N_{\text{KCs}} + N_{\text{prerequisites}} - 1)$ query variables.

The inference task is mobilized plenty of times in the E-PRISM framework. It must be performed several times in E-PRISM for the framework to provide the main ITS functionalities. Because exact inference is NP-hard, its intractability is inevitable. At least, it cannot scale for more complex tasks such as parameter learning.

Exact inference running time

The number of query variables is directly related to the size of the state space and depends on N_{KCs}, $N_{\text{prerequisites}}$, and N_{traces}. We study the computational time required to exactly infer the evolution of the knowledge state of a learner over time to apprehend the difficulties of exact inference in E-PRISM better. More particularly, we compute the running time of a learner's knowledge state diagnosis over time for a growing number of learner traces on domain models with different sizes. From a mathematical perspective, we compute the probability $P(\mathbf{y} \mid \mathbf{e})$ of the whole knowledge state $\mathbf{y} = \mathbf{x} \setminus \mathbf{e}$, where $\mathbf{x}$ regroups the DBN variables and $\mathbf{e}$ is the evidence from the learner traces.

We introduce four different domain models to understand better the dependence of the computation complexity with both the number of KCs in the domain model and the number of transactions taken into account for the inference task.

- The first domain model consists of two independent KCs $\mathfrak{A}$ and $\mathfrak{B}$;
- The second domain model consists of two KCs $\mathfrak{A}$ and $\mathfrak{B}$ related with each other with the prerequisite relationship $\mathfrak{A} \rightarrow \mathfrak{B}$;
- The third domain model consists of three KCs $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ such that there are the prerequisite relationships $\mathfrak{A} \rightarrow \mathfrak{C}$ and $\mathfrak{B} \rightarrow \mathfrak{C}$;
- The fourth domain model is composed of four KCs $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, and $\mathfrak{D}$ such that $\mathfrak{A} \rightarrow \mathfrak{D}$, $\mathfrak{B} \rightarrow \mathfrak{D}$, and $\mathfrak{C} \rightarrow \mathfrak{D}$.

For each domain model configuration, we instantiate an E-PRISM learner model. With each of them, we compute the probability distribution $P(\mathbf{y} \mid \mathbf{e})$ with exact inference on the E-PRISM learner model. We represent in Figure 5.1 the running time of the exact inference computation on each domain model as a function of the number of learner transactions considered. The calculations are performed on a single core of Apple's M1 chip (3.2 GHz with 16 Go RAM).

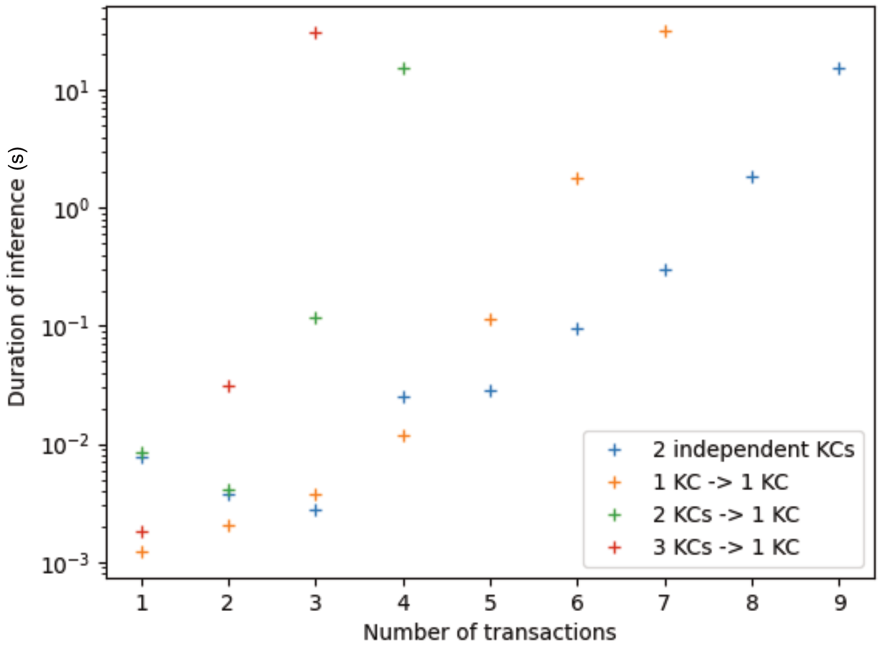


Figure 5.1: Running time (in seconds) taken to exactly infer $P(\mathbf{y}|\mathbf{e})$, as a function of the number of transactions taken into account in the evidence $\mathbf{e}$. Each color corresponds to a different domain model. The vertical axe is log-linear, as the running time of exact inference is an exponential function of the number of learner transactions.

Figure 5.1 shows that the running time of exact inference exponentially grows with the number of transactions, whatever the domain model. Because the number of query variables directly depends on the number of transactions, this result sounds logical. We also observe the running time of exact inference is greater on more complex domain models for the same number of transactions. The study of the running time of the exact inference computation

is a glimpse at the exact inference intractability. As the number of KCs and prerequisite relationships grows, exact inference becomes quickly intractable, even for a low number of learners' transactions with the system.

The E-PRISM framework must be capable of performing these calculations in a scalable way to provide the main ITS functionalities. Indeed, the computation of these query probabilities is often included in more complex procedures, such as model learning. Therefore, exact inference is not suitable for domain models of substantial size. Alternative inference methods must be implemented.

5.1.3 Gibbs sampling for approximate inference in E-PRISM

Given that exact inference quickly becomes intractable, alternative methods for approximate inference are necessary. This work explores the use of particle-based approximate inference. Bayesian networks are ideal models for Monte-Carlo methods, as shown in Chapter 3. This section provides a complete overview of classic Gibbs sampling in the E-PRISM learner model.

Gibbs sampling in BNs

Gibbs sampling is an easy-to-implement and comprehensive technique to approximate the computation of probability queries. In a nutshell, Gibbs sampling is a kind of Markov Chain Monte-Carlo (MCMC) method, introduced by Geman and Geman [GG84]. It allows sampling from a complex distribution that can be fully described with a set of conditional probability distributions. Gibbs sampling is described in Algorithm 3.3.

We recall the principle of the Gibbs sampling algorithm. Let be a BN composed of random variables $\mathbf{x} = (x_1, ..., x_n)$. Gibbs sampling in this BN is an iterative process. First, it sets the initial state of network variables in a random state $\mathbf{x}^{(0)} = (x_1^{(0)}, ..., x_n^{(0)})$. At each iteration k of the Gibbs sampling, the state of the BN variables $\mathbf{x}^{(k)} = (x_1^{(k)}, ..., x_n^{(k)})$ is determined from the state $\mathbf{x}^{(k-1)}$ of the BN variables at the $(k-1)$-th iteration. Generally, the BN variables are randomly or topologically ordered. Then, given a specific order, the j-th variable of the BN at the step k of the Gibbs sampling $x_j^{(k+1)}$ is sampled from the conditional probability distribution depicted in Equation 5.3.

$$P(x_j \mid \mathbf{x}_{-j}^{(k)}) = P(x_j \mid x_1^{(k)}, ..., x_{j-1}^{(k)}, x_{j+1}^{(k-1)}, ..., x_n^{(k-1)}) \tag{5.3}$$

The factorization induced by BN structures makes Gibbs sampling particularly well-adapted to BNs. The condition in the CPD in Equation 5.3 can be simplified. $\forall x_j \in \mathbf{x}$, x_j only depends on its Markov blanket, so the condition can be reduced to it. Gibbs sampling in a Bayesian network then consists of successive samplings of the network variables from the limited study of their Markov blankets.

In practice, a burn-in period is integrated into the Gibbs sampling implementation. The samples obtained from the first iterations of the Markov chain are not considered, as the chain has not reached the target stationary distribution yet. This MCMC stationary distribution is the distribution we want to sample from, and it can be reached only after several iterations of the Gibbs sampling algorithm. Nevertheless, the efficient length of the burn-in period cannot be determined beforehand [KF09]. Also, a sampling period is sometimes considered. Integrating a sampling should remove the correlation of successive samples to guarantee they are Independent and Identically Distributed (i.i.d.). The Gibbs sampling indeed assumes i.i.d. samples to ensure the convergence of the Markov chain.

We can represent the process of Gibbs sampling by representing the state space of the whole distribution on a 2D plane. Figure 5.2 illustrates the convergence of the Gibbs sampling algorithm. Because the whole state space is wider than the target distribution space, it represents the potential need for a burn-in period denoted M. The first step of Gibbs sampling consists in reaching the target distribution space. Then, the particle can move from the states of the target distribution. Still, reaching the target distribution space is not sufficient for ensuring the convergence of the Gibbs sampling process, as depicted in Figure 5.2.

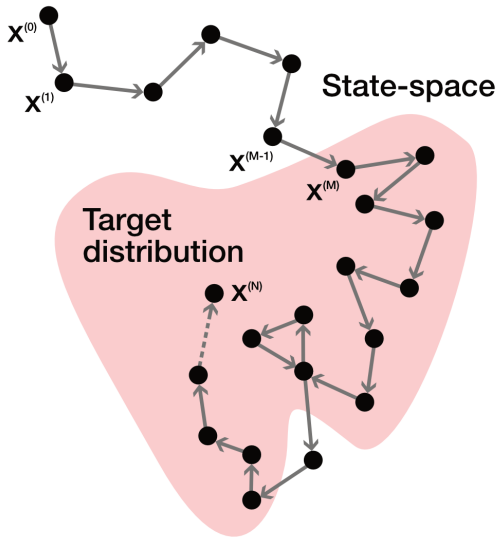


Figure 5.2: Graphical representation of the Gibbs sampling algorithm if one supposes that the state space of the distribution can be represented in a 2D plane. M is the burn-in period, and N is the length of the Gibbs sampling. The target distribution is approximated with $N - M$ samples $(\mathbf{x}^{(M)}, ..., \mathbf{x}^{(N)})$.

Convergence of Gibbs sampling

The convergence of the Gibbs sampler entails its capacity to reach the stationary distribution. We have seen in Section 3.2 that the Markovian process should be irreducible to ensure convergence of the Gibbs sampling. In other words, there must exist a strictly positive transition

probability between each pair of states of the distribution. A working Gibbs sampling should visit every state with a positive probability in the target distribution. However, the time taken for the chain to "mix" is difficult to predict, as there is no mathematical expression for the convergence speed.

To assert the convergence of the Gibbs sampling procedure, the sampled distribution can be compared with the distribution computed from exact inference. The limits of this approach for evaluating the convergence of Gibbs sampling are trivial. Suppose approximate inference techniques are necessary to determine the distribution. In that case, exact inference is probably too complex, making it impossible to compare the sampled distribution with the intractable exact inference. In our case, we recall that exact inference is only tractable for a limited number of learner traces.

Köller et al. propose another approach to determine the convergence of Gibbs sampling. They study multiple chains aiming to sample the same distribution, and they compare them afterward, as they should lead to similar estimators with comparable variance [KF09]. The resulting metrics represent the estimate of a given function f and they entirely rely on the chosen function f. For instance, in our case, the functions f would be the indicator functions for each combination of states. Similarly to the first technique for asserting convergence, this technique would not be tractable for many learner traces, as the number of functions to check would be too high. Therefore, we can only check the convergence of the Gibbs sampling algorithm for a limited number of learner traces.

Deterministic CPDs as sources of reducibility

For each kind of variable in the DBN of the E-PRISM learner model, we detail the Markov blanket and the values of the CPD required for updating its state during Gibbs sampling in Appendix A. The whole CPD $P(x_i|\mathbf{x}_{-i}^{(k)})$ can be simplified for each $x_i \in \mathbf{x}$. The condition in the CPD can be restricted to the Markov blanket for each variable of the DBN.

Gibbs sampling in the DBN of the E-PRISM learner model does not need any complex computation. The probability of any BN variable, given its Markov blanket, is already stored in the CPDs ruling the network. Furthermore, there are additional constraints in the E-PRISM learner model CPDs thanks to the deterministic functions that rule the DBN. This results in a further simplified Gibbs sampling procedure. Nevertheless, the ICI-based CPDs in the E-PRISM learner model also imply that the Gibbs sampling procedure is reducible. We can spot that some transition probabilities between two positive states of the BN equal zero in Appendix A. This doesn't mean that Gibbs sampling does not converge, but it reveals that its convergence is not ensured.

We study the convergence of Gibbs sampling for approximate inference in the E-PRISM learner model. The convergence is determined from the computation of the Kullback-Leibler (KL) divergence between the approximately-inferred and exactly-inferred probability

distributions from evidence. We suppose the domain model of Example 4.2.1 that consists of three KCs $(\mathfrak{A}, \mathfrak{B}, \mathfrak{C})$ with prerequisite relationships $\mathfrak{A} \rightarrow \mathfrak{C}$ and $\mathfrak{B} \rightarrow \mathfrak{C}$. We define the inference task as the computation of the probability distribution $P(\mathbf{y} \mid \mathbf{e})$, with $\mathbf{e} = (C^0 = 0, A^1 = 1, A^2 = 1)$. Each approximated inference task is performed on a single core of Apple's M1 chip (3.2 GHz with 16 Go of RAM).

In the first place, we suppose there is no burn-in period. We plot the convergence of the inference task as a function of the number of Gibbs iterations in Figure 5.3. We represent box plots of the KL divergence between probability distributions computed from approximate and exact inference. Each box plot emphasizes the distribution of 100 runs of the Gibbs sampling estimator, consisting of all samples generated during Gibbs sampling.

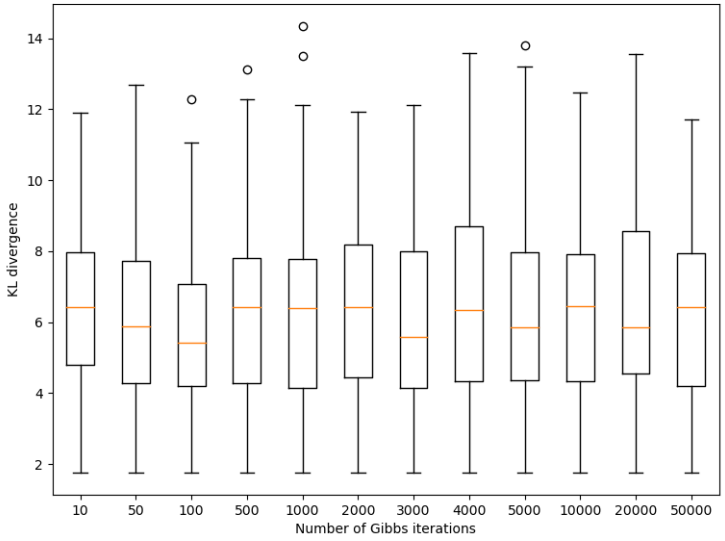


Figure 5.3: Kullback-Leibler divergence between the approximately-inferred probability distribution $P_{\text{Gibbs}}(\mathbf{y}|\mathbf{e})$ and the exactly inferred probability distribution $P_{\text{exact}}(\mathbf{y}|\mathbf{e})$ as a function of the number of Gibbs iterations N_{Gibbs}, for $\mathbf{e} = (C^0 = \neg c, A^1 = +a, A^2 = +a)$.

We can observe that the value of the KL divergence is the same, whatever the number of Gibbs iterations. The Gibbs sampler seems to stick in a particular state of the target distribution. The number of Gibbs sampling iterations does not affect the existence of this potential well. Warnings about the possible non-convergence of reducible Markovian processes seem to be fully realized.

Secondly, we study the impact of the burn-in period on the convergence of the Gibbs sampling approximate inference. We suppose $N_{\text{Gibbs}} = 10\ 000$. We have chosen a high number of Gibbs iterations to increase the chances of convergence. We represent the KL divergence between exact and approximate inference computations as a function of M, the burn-in period, in Figure 5.4.

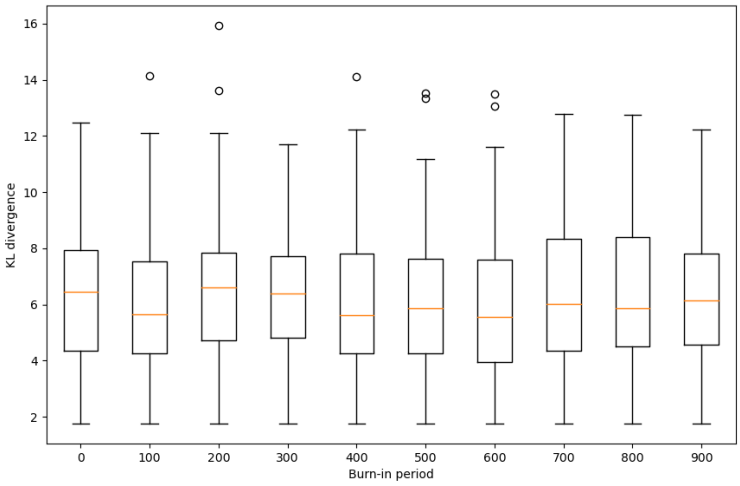


Figure 5.4: Kullback-Leibler divergence between the approximately-inferred probability distribution $P_{\text{Gibbs}}(\mathbf{y}|\mathbf{e})$ and the exactly inferred probability distribution $P_{\text{exact}}(\mathbf{y}|\mathbf{e})$ as a function of the burn-in period M, for $\mathbf{e} = (C^0 = \neg c, A^1 = +a, A^2 = +a)$.

Using a burn-in period is insufficient for converging Gibbs sampling, according to Figure 5.4. As reported by Köller [KF09], the burn-in period cannot be determined beforehand, so we cannot ensure that Gibbs sampling is not converging in E-PRISM for greater values of N_{Gibbs} and M. However, it is important to remember that the inference task is a crucial component in larger E-PRISM processes, such as parameter learning, which repeatedly utilize it. The chosen number of Gibbs iterations is already high in the experiment reported in Figure 5.4. Excessive values for N_{Gibbs} and M would make these procedures intractable anyway.

The Gibbs sampling non-convergence is a major issue when using the E-PRISM learner model. The inference task is indeed mandatory for model learning. When tweaking around Gibbs sampling on toy examples, we observe that some sets of variables are stuck in local equilibria, leading to not convergent Gibbs sampling.

Example 5.1.4 (Potential well in Gibbs sampling)**.** *Suppose a domain model with two knowledge components $\mathfrak{A}$ and $\mathfrak{B}$ related to each other by the prerequisite relationship $\mathfrak{A} \to \mathfrak{B}$. We represent a part of the DBN of the E-PRISM learner model with two transactions from the learner in Figure 5.5.*

Suppose the vertices $T^1_{\mathfrak{B}}$, $Z^1_{\mathfrak{B},\mathfrak{A}}$, and B^1 have reached the state depicted in Figure 5.5. The Gibbs sampling is stuck in a local equilibrium state which leads to a biased sampling and a wrong estimator of the probability distribution. Whatever the state of the other variables of the network, the probability that $T^1_{\mathfrak{B}}$, $Z^1_{\mathfrak{B},\mathfrak{A}}$, and B^1 change their state during Gibbs sampling is zero.

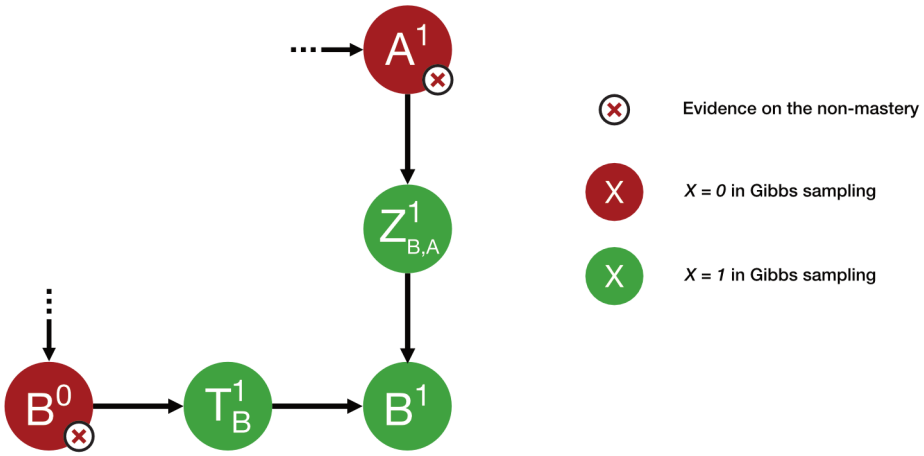


Figure 5.5: Example of a situation of a potential well during the Gibbs sampling. The evidence is composed of $\mathbf{e} = (B^0 = \neg b, A^1 = \neg a)$ and the Gibbs sampling has reached the state $(T^1_{\mathfrak{B}} = +t, Z^1_{\mathfrak{B},\mathfrak{A}} = +z, B^1 = +b)$, which is a local equilibrium.

The non-convergence issue of Gibbs sampling can be explained by returning to its 2D representation. We can represent the presence of potential wells as positions where a movement to other adjacent positions is not allowed. We provide the graphical explanation of the non-convergence of Gibbs sampling in the presence of constraints such as deterministic functions in Figure 5.6.

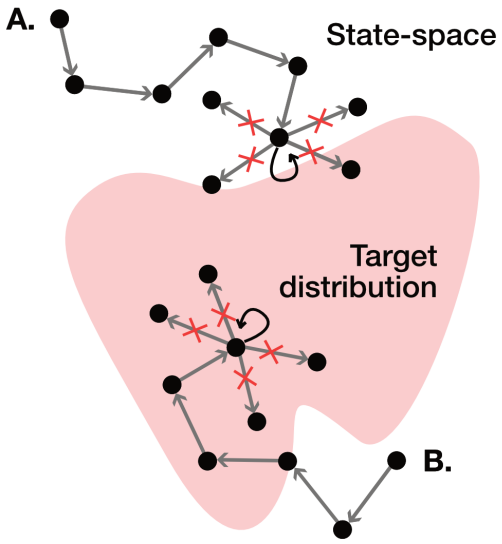


Figure 5.6: Graphical representation of the Gibbs sampling non-convergence when the target distribution is supposed to be represented in a 2D plane. In situation A, the Markovian process does not even have the time to reach the stationary distribution before being trapped in a potential well. In situation B, the Markovian has reached the stationary distribution but is trapped in one of the stationary distribution states, providing an incorrect description of the whole distribution.

We represent two typical situations in which a potential well can occur. In situation A, the potential well prevents the Markovian process from reaching the stationary distribution. In situation B, the potential well is spotted once the stationary distribution is reached but makes the approximate inference to overrepresent one of the states of the target distribution.

Importance sampling

Some strategies can be deployed to allow convergence for approximate inference on BNs with deterministic CPDs. First, it is possible to regroup some BN variables to get back to an irreducible Markovian process. This solution is difficult to apply in practice, as the purpose of the E-PRISM learner model was to grant higher interpretability to the model parameters. Otherwise, we can introduce a positive real number $\gamma > 0$ such that $\gamma \ll 1$. The probability of illegal events, which are events with zero probability, is set to γ. This strategy leads to a theoretical irreducibility of the Markovian process. Sheehan has demonstrated that the convergence of the Gibbs sampling can be achieved with this technique if illegal configurations, sampled states not following the network constraints, are deleted when computing the expectation on the samples [She00].

The general approach for solving the reducibility of a Markovian process is the importance sampling approach, presented in Chapter 3. Instead of sampling the initial probability density f, one can sample a well-chosen probability density q leading to an irreducible Markovian process. The estimator generated from Gibbs sampling would then correspond to the one described in Equation 5.4.

$$\frac{1}{N}\sum_{l=1}^{N} w(x(t_l))F(x(t_l)) \text{ with } w(x) = f(x)/q(x) \tag{5.4}$$

The estimator generated from Gibbs sampling from probability density q is the expectation $E_q[w(x)F(x)]$. We can prove that $E_q[w(x)F(x)] = E_f[F(x)]$. The expectation from the samples generated from Gibbs sampling from probability density q is the same as the expectation from Gibbs sampling from probability density f. Note that the γ approach is, in practice, a special case of importance sampling, where the probability density q is the one described in Equation 5.5.

$$q(x) = \frac{f(x) + \epsilon}{1 + \epsilon} \text{ with } \epsilon > 0,\ \epsilon \ll 1 \tag{5.5}$$

We now focus on how using importance sampling influences the convergence of Gibbs sampling. We keep performing Gibbs sampling inference on the E-PRISM learner model introduced previously. We represent the convergence of Gibbs sampling for a number of Gibbs iterations $N_{\text{Gibbs}} = 10\ 000$ and a burn-in period $M = 9\ 000$. We study the KL divergence as a function of the γ value employed during the sampling in Figure 5.7.

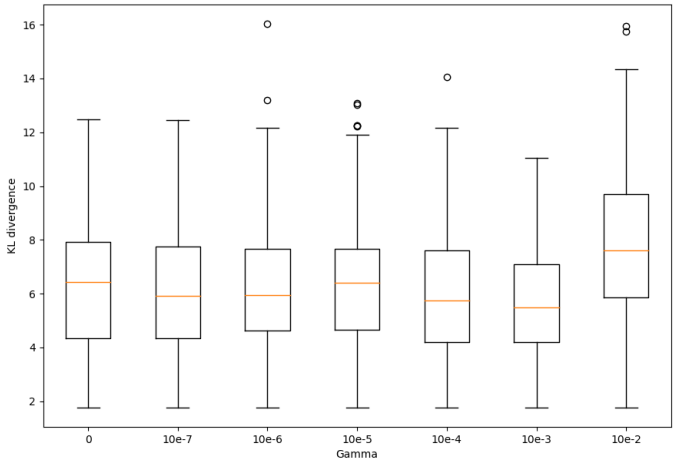


Figure 5.7: Kullback-Leibler divergence between the approximated inferred probability distribution $P_{\text{approx}}(\mathbf{y}|\mathbf{e})$ with importance sampling and the exactly inferred probability distribution $P_{\text{exact}}(\mathbf{y}|\mathbf{e})$ as a function of the gamma parameter γ, for $\mathbf{e} = (C^0 = \neg c, A^1 = +a, A^2 = +a)$.

As stated before, introducing a small positive real number to delete the possibility of zero probability is a solution. However, we observe that the potential wells still exist, and it is complex to prevent the sampler from being stuck in them. Because the γ value is close to zero, the chance of leaving the well is subject to high variance. In practice, for MCMC inference in the E-PRISM learner model, we observe that the estimator sometimes gives the correct distribution. Still, it often shows disproportionately represented states of the distribution. We even observe that for high γ values, the KL divergence increases. Because the main goal of inference in this work is the parameter learning task, using the classic Gibbs sampling algorithm for approximate inference in E-PRISM is not viable.

5.1.4 Blocking Gibbs sampling

Previous sections have presented the issues of the various inference techniques in the E-PRISM learner model. In this section, we introduce a new approximate inference technique based on Blocking Gibbs Sampling (BGS) in the context of BNs with deterministic CPDs. This technique is designed to address the convergence issues spotted with Gibbs sampling. BGS, first introduced by Jensen et al. [JKK95], involves considering blocks of variables instead of individual variables. It speeds up the sampling convergence of the MCMC. The bigger the blocks, the faster the convergence of sampling [AGP91]. The size of the variable blocks chosen for Blocking Gibbs sampling is only limited by the computational capacity of the machine used for sampling.

Blocks of variables as irreducible elements of the Markovian process

Instead of sampling the variables of the learner model DBN one by one, we then sample blocks of variables simultaneously. We want to benefit from the structure of the E-PRISM learner model, in particular, the ICI-based CPDs that compose the DBN. The purpose of this approach is not to speed up the convergence of the sampler, but it is simply to allow convergence in a situation where it is complicated to get. Thus, we use meaningful blocks rather than the largest possible blocks.

Each block is associated with a KC and a timeslice. Blocks are indexed with time because of the dynamic nature of the learner model. We differentiate initial, transition, and final blocks, depending on the corresponding timeslice. The DBN of the learner model can be expressed as the concatenation of these blocks. We represent them in Figures 5.8a, 5.8b and 5.8c. In particular, we highlight the Markov blanket of these blocks.

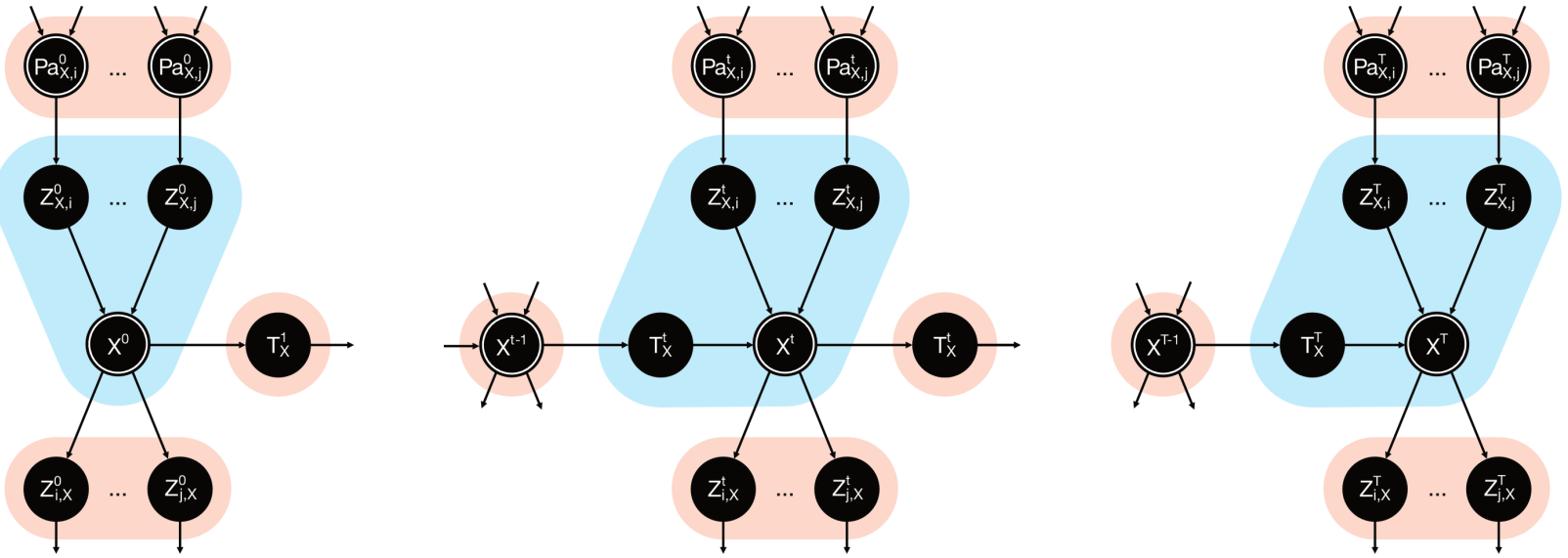
The initial blocks represent the initial timeslice of the unrolled DBN of the E-PRISM learner model. There is an initial block per KC in the domain model. $B^0_{\mathfrak{X}}$, the initial block related to the KC $\mathfrak{X}$ is composed of the variables X^0, and $\{Z^0_{\mathfrak{X},i}\}_i$ if $\mathfrak{X}$ has parent KCs. Its Markov blanket is composed of variables $T^1_{\mathfrak{X}}$, $\{Pa^0_{\mathfrak{X},i}\}_i$ if $\mathfrak{X}$ has parent KCs, and $\{Z^0_{j,\mathfrak{X}}\}_j$ if $\mathfrak{X}$ has child KCs.

The final blocks are blocks related to the last timeslice of the unrolled DBN. There is a final block per KC in the domain model. We denote T, the index of the last timeslice. $B^T_{\mathfrak{X}}$, the final block related to $\mathfrak{X}$ is composed of the variables X^T, $T^T_{\mathfrak{X}}$, and $\{Z_{\mathfrak{X},i}\}_T$ if $\mathfrak{X}$ has parent KCs. Its Markov blanket is the set of variables X^{T-1}, $\{Pa^T_{\mathfrak{X},i}\}_i$ if $\mathfrak{X}$ has parent KCs, and $\{Z^T_{j,\mathfrak{X}}\}_j$ if $\mathfrak{X}$ has child KCs.

The transition blocks represent all the other timeslices of the unrolled DBN. For each t such that $0 < t < T$, there is a transition block per KC in the domain model. $B^t_{\mathfrak{X}}$, the transition block related to $\mathfrak{X}$ is composed of the variables X^t, $T^t_{\mathfrak{X}}$, and $\{Z_{\mathfrak{X},i}\}_t$ if $\mathfrak{X}$ has parent KCs. Its Markov blanket is the set of variables X^{t-1}, $T^{t+1}_{\mathfrak{X}}$, $\{Pa^t_{\mathfrak{X},i}\}_i$ if $\mathfrak{X}$ has parent KCs, and $\{Z^t_{j,\mathfrak{X}}\}_j$ if $\mathfrak{X}$ has child KCs.

Implementation of approximate inference in the learner model with Blocking Gibbs sampling

First, we suppose the blocks of variables to be topologically ordered. Given the structure of the blocks, the topological order is given by the prerequisite relationships' ordering. We denote $\succ$, the topological order of the blocks. We can prove that $\forall$ KC $\mathfrak{X}, \forall t, t'$ s.t. $t < t', B^t_{\mathfrak{X}} \succ B^{t'}_{\mathfrak{X}}$. We also have for any pair of KCs $\mathfrak{X}, \mathfrak{X}'$, if $\mathfrak{X} \to \mathfrak{X}'$, then $\forall t, B^t_{\mathfrak{X}} \succ B^t_{\mathfrak{X}'}$. .



(a) Initial block **(b)** Transition block **(c)** Final block

Figure 5.8: Graphical representation of the blocks associated with the KC $\mathfrak{X}$ in the learner model of the E-PRISM framework. The block consists of the nodes circled in blue. The Markov blanket of the transition block is circled in orange.

We recall that the purpose of approximate inference is to compute the probability distribution $P(\mathbf{x}, \mathbf{y}|\mathbf{e})$, where $\mathbf{e}$ is the evidence $(e_1, ...e_T)$, $\mathbf{x}$ is the set of variables associated with the hidden or non-observed vertices of the network, and $\mathbf{y}$ is the set of variables associated with the observed vertices of the network.

Blocking Gibbs sampling consists of the same steps as classic Gibbs sampling. It iterates N_{Gibbs} times over every block in the DBN in the topological order and generates a new sample from the target distribution. The initial state of the network in the sampling process is set up randomly. It must still verify the deterministic functions that compose the DBN. Then, in the manner of the Gibbs sampling, the blocks are successively browsed following the topological order.

Only the state of the Markov blanket of a variable must be known when updating its state during the Gibbs sampling. Similarly, one must know the Markov blanket of a block to update the state of the variables that compose it in Blocking Gibbs sampling. The states of each variable of a block are updated simultaneously according to the state of the block's Markov blanket by sampling from the conditional probability $P(B^t_{\mathfrak{X}} \mid \mathcal{MB}(B^t_{\mathfrak{X}}), \mathbf{e})$, with $\mathcal{MB}(B^t_{\mathfrak{X}})$ the Markov blanket of $B^t_{\mathfrak{X}}$. The computation is done with exact inference. Exact inference can be easily performed on described blocks, as the BNs representing them and their Markov blanket are small enough, as depicted in Figure 5.8.

We detail the implementation of Blocking Gibbs sampling for approximate inference in the learner model in Algorithm 5.1. This novel approach can be applied to any system modeled with a BN composed of ICI-model CPDs.

Convergence of Blocking Gibbs sampling for approximate inference

As we did previously with Gibbs sampling, we examine the convergence of Blocking Gibbs sampling in the context of ICI-based BNs through several E-PRISM practical examples. We focus on studying the KL divergence as a function of the number of Gibbs iterations N_{Gibbs} and the burn-in period M. We will see that the blocking strategy eliminates the presence of potential wells. Consequently, importance sampling will not be necessary.

We use the same domain model from Example 4.2.1 that consists of three KCs $(\mathfrak{A}, \mathfrak{B}, \mathfrak{C})$ with prerequisite relationships $\mathfrak{A} \rightarrow \mathfrak{C}$ and $\mathfrak{B} \rightarrow \mathfrak{C}$. We examine the probability distribution $P(\mathbf{y} \mid \mathbf{e})$ with $\mathbf{e} = (C^0 = 0, A^1 = 1, A^2 = 1)$. We compare it to the exact inference with the computation of the KL divergence between the approximately-inferred and exactly-inferred distributions. The calculations are still performed on a single core of Apple's M1 chip (3.2 GHz with 16 Go RAM).

Initially, we do not consider a burn-in period, so the estimator is based on all the samples generated during BGS. The KL divergence between the BGS approximation and the exact inference is plotted in Figure 5.9 as a function of the number of Gibbs iterations. Unlike Gibbs sampling, the KL divergence decreases exponentially with the number of Gibbs iterations.

Algorithm 5.1: Blocking Gibbs sampling in an ICI-based Bayesian Network $\mathcal{B}$

Data: ICI-based Bayesian network $\mathcal{BN}$ with variables $X_1, ..., X_N$, N_{Gibbs}, M
Result: $\mathbf{x} = (\mathbf{x}^{(M)}, ..., \mathbf{x}^{(N_{\text{Gibbs}}-M)})$

$\mathcal{B}_1, ..., \mathcal{B}_k \leftarrow$ GetBlocksTopologicalOrder($\mathcal{BN}$)
$(x_1, ..., x_N) \leftarrow$ RandomInitialization($\mathcal{BN}$)
for $r = 1$ *to* N_{Gibbs} **do**
  for $j = 1$ *to* k **do**
    MB $\leftarrow$ GetMarkovBlanket($\mathcal{B}_j$)
    $\mathbf{e} \leftarrow (\mathbf{b}_1^{(r)}, ..., \mathbf{b}_{j-1}^{(r)}, \mathbf{b}_{j+1}^{(r-1)}, ..., \mathbf{b}_k^{(r-1)})_{|\text{MB}}$
    $\mathbf{b}_j^{(r)} \propto P(\mathcal{B}_j \mid \mathbf{e})$ ▷ with exact inference
  end
  $\mathbf{x}^{(r)} \leftarrow (\mathbf{b}_1^{(r)}, ..., \mathbf{b}_k^{(r)})$
  if $r > M$ **then**
    Add $\mathbf{x}^{(r)}$ to $\mathbf{x}$
  end
end

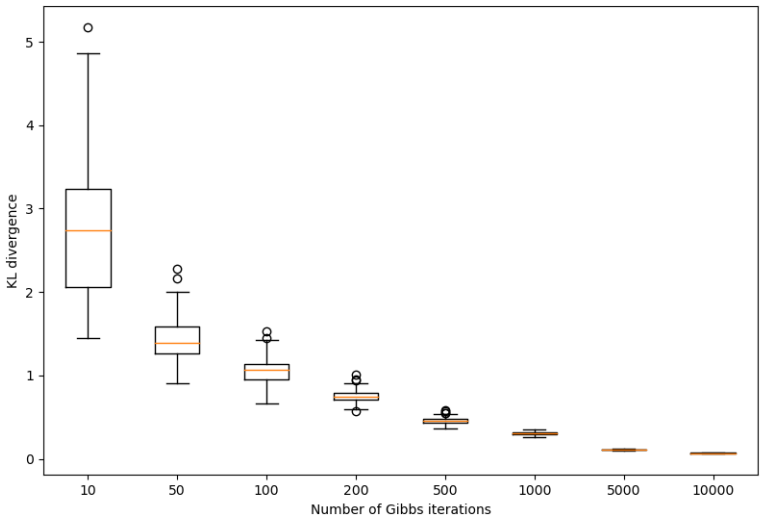


Figure 5.9: KL divergence between the BGS approximately-inferred probability distribution $P_{\text{approx}}(\mathbf{y}|\mathbf{e})$ and the exactly-inferred probability distribution $P_{\text{exact}}(\mathbf{y}|\mathbf{e})$ as a function of the number of Gibbs iterations N_{Gibbs}, for $\mathbf{e} = (C^0 = \neg c, A^1 = +a, A^2 = +a)$.

Additionally, we investigate the effect of the burn-in period by assuming $N_{\text{Gibbs}} = 1000$. The BGS convergence is shown as a function of the burn-in period in Figure 5.10. It indicates no significant effect on the convergence of BGS using a burn-in period. Hence, it can be concluded that using a burn-in period is not necessary for Blocking Gibbs sampling in the learner model.

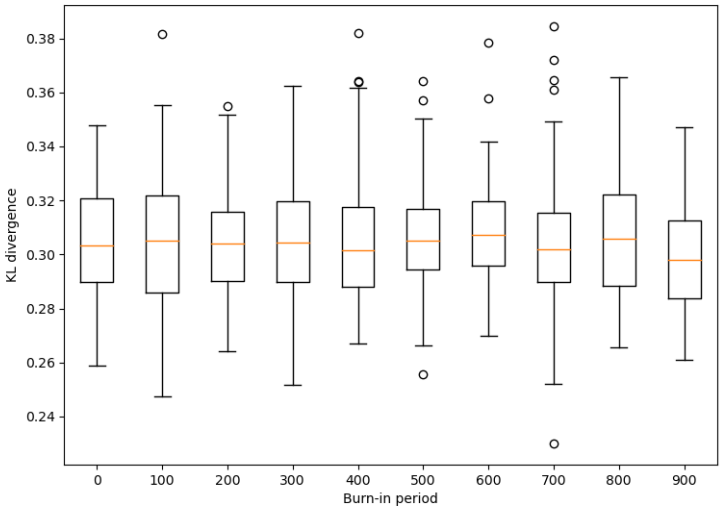


Figure 5.10: KL divergence between the BGS approximated inferred probability distribution $P_{\text{approx}}(\mathbf{y}|\mathbf{e})$ and the exactly inferred probability distribution $P_{\text{exact}}(\mathbf{y}|\mathbf{e})$ as a function of the burn-in period M, for $\mathbf{e} = (C^0 = \neg c, A^1 = +a, A^2 = +a)$.

Running time of Blocking Gibbs sampling

We now examine the running time of the approximate inference by Blocking Gibbs sampling. We use several domain model configurations similar to those introduced in Section 5.1.2. The computations are still performed on a single core of Apple's M1 chip (3.2 GHz with 16 Go RAM). It is the same as the one used for computing the running time of the exact inference technique. Thus, we can compare the results of the two experiments.

- Domain model 1: two independent KCs $\mathfrak{A}$ and $\mathfrak{B}$;
- Domain model 2: two KCs $\mathfrak{A}$ and $\mathfrak{B}$ such that $\mathfrak{A} \to \mathfrak{B}$;
- Domain model 3: three KCs $\mathfrak{A}$, $\mathfrak{B}$, and $\mathfrak{C}$ such that $\mathfrak{A}, \mathfrak{B} \to \mathfrak{C}$;
- Domain model 4: four KCs $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, and $\mathfrak{D}$ such that $\mathfrak{A}, \mathfrak{B}, \mathfrak{C} \to \mathfrak{D}$.

We recall that the target probability query is the probability $P(\mathbf{y}|\mathbf{e})$, with $\mathbf{y}$ the set of query variables and $\mathbf{e}$ the set of evidence variables.

First, we look at the running time as a function of the number of transactions expressed in the evidence $\mathbf{e}$. The number of Gibbs iterations for Blocking Gibbs sampling is fixed

at $N_{\text{Gibbs}} = 1000$. This value was chosen due to its good performance in the previous experiment. The BGS running time is represented in Figure 5.11 for each domain model configuration listed above.

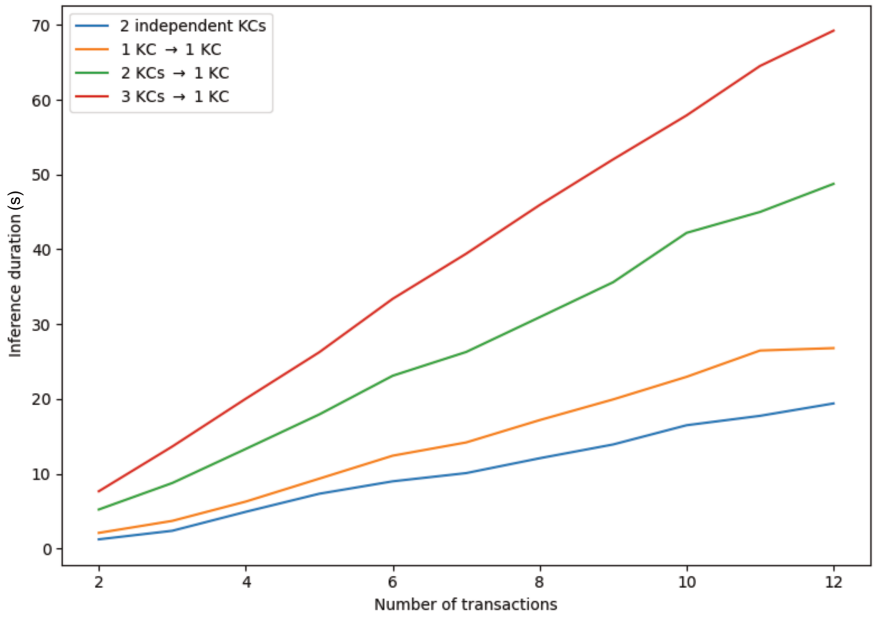


Figure 5.11: Representation of the computational time (in seconds) of the Blocking Gibbs sampling approximate inference of $P(\mathbf{y}|\mathbf{e})$ as a function of the number of learner transactions contained in $\mathbf{e}$ for several domain model configurations. The number of Gibbs iterations is fixed to $N_{\text{Gibbs}} = 1000$, and the burn-in period is set to $M = 0$.

In Figure 5.11, the running time of the approximated inference by Blocking Gibbs sampling seems to grow linearly with the number of learner transactions for all configurations of the domain model. The slope of the curve seems to be linearly dependent on the number of KCs and the number of prerequisite relationships. We also observe that the running time of exact inference is much longer than that of approximated inference from several learner transactions in $\mathbf{e}$ when comparing these results with those in Figure 5.1. Nevertheless, we still remark that the exact inference is more efficient for small numbers of transactions than approximated inference. This can be explained by the fact that approximated inference is fully developed in `Python`, while the exact inference relies on `aGrUm`, which is a `C++`-based library.

In the second place, we focus on the second domain model configuration. We recall it supposes two KCs $\mathfrak{A}$ and $\mathfrak{B}$ with $\mathfrak{A} \rightarrow \mathfrak{B}$. We study the influence of the number of Gibbs iterations N_{Gibbs} on the computational time of BGS inference in a learner model related to this domain model. Figure 5.9 has shown N_{Gibbs} greatly influences the accuracy of the approximated inference. We then represent in Figure 5.12 the BGS inference running time as a function of the number of transactions considered in $\mathbf{e}$ for several values of the number of Gibbs iterations in the Blocking Gibbs sampling approximated inference.

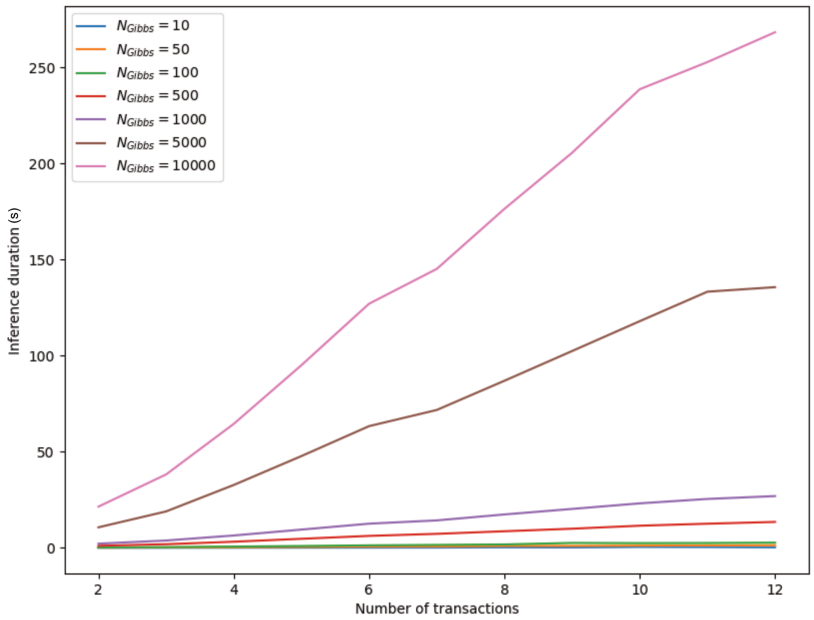


Figure 5.12: Representation of the computational time of the BGS approximate inference of $P(\mathbf{y}|\mathbf{e})$ as a function of the number of learner transactions contained in $\mathbf{e}$ for different values of the number of Gibbs iterations N_{Gibbs}.

In Figure 5.12, we can figure out that the running time seems linear with the number of learner traces, no matter the number of Gibbs iterations N_{Gibbs}. We observe that the slope of the computational time is proportional to N_{Gibbs}.

Using Blocking Gibbs sampling for approximate inference is more applicable in practice than using exact inference. Indeed, we must keep in mind that the inference task is an element of larger processes, such as parameter learning. Many learners' transactions and Gibbs iterations would result in intractable parameter learning, as the inference task is mobilized thousands of times.

5.2 Updating the E-PRISM learner model

The parameter learning procedure is a critical component of the E-PRISM learner model, as it allows the system to update its knowledge about the learner over time based on evidence. As explained in Chapter 3, the learning procedure for a Bayesian network can be decomposed into two tasks: parameter learning and structure learning. Basically, structure learning consists of an iterative process of parameter learning. Therefore, we only focus on parameter learning in our work.

Thanks to the great interpretability of the E-PRISM parameters described in Section 4.2, the parameter learning procedure can provide valuable insights into the prerequisite structure of the domain knowledge. E-PRISM can provide a more comprehensive understanding of the domain knowledge structure by analyzing the KC relationships through the values of its interpretable parameters. It gives insights into the prerequisite structure in addition to insights into learning and forgetting phenomena. These insights are valuable for developing effective learning experiences and identifying areas where additional support may be needed.

In this section, we explore the parameter learning procedure for the E-PRISM learner model. We presented the structure of the learner model in Section 4.2 and introduced a DBN modeling learners' knowledge over time. As explained in Section 4.2.4, DBNs with ICI-model CPDs imply the presence of latent variables in the network. Consequently, the dataset relating the learner traces cannot be complete, as model variables won't be observed whatever the context. The EM algorithm is the most common procedure for parameter learning in the presence of non-observed variables in the learning dataset, as stated in Section 3.3.

First, we look at the principles of the EM algorithm applied to the DBN in the learner model. We explain how the data from learner traces must be preprocessed to allow the computations in the E and M steps. Then, we introduce stochastic approaches for parameter learning. Finally, we analyze the results of the parameter learning algorithm on synthetic data.

5.2.1 Available data for E-PRISM parameter learning

Chapter 3 only considers classic Bayesian networks for parameter learning. However, the E-PRISM framework relies on a Dynamic Bayesian Networks (DBNs) representing a whole family of BNs with a similar structure. From one data entry to another, the number of variables in a DBN can be very different, as the unrolled DBN can be of any length. Consequently, the first step is to preprocess the data for parameter learning with DBN.

Former datasets

In our case, a learner can do as many transactions as she wants. For N_{traces} transactions done by a learner, the dataset relating to the learner transactions will have N_{traces} rows. Each contains information on one of the N_{traces} transactions, such as the resource on which the transaction is collected, the associated KC, the answer correctness, and the transaction's

timestamp. However, these data don't fit the structure of the DBN. The corresponding DBN employed for inferring the learner's knowledge state over time from these traces is composed of N_{traces} timeslices. We must transform the N_{traces} rows of the initial dataset into a single entry that relates the state of the N_{traces} timeslices of the DBN. We detail the procedure of the dataset transformation in the following.

Example 5.2.1 (Former dataset)**.** *Suppose a domain model is composed of the KCs* $\mathfrak{X}_1$, $\mathfrak{X}_2$, *and* $\mathfrak{X}_3$. *Each of them is associated with at least one exercise. We introduce a toy example of a possible dataset obtained from learner traces of two learners on resources associated with this domain model in Table 5.1. It comprises the following characteristics: temporality through dataset index, learner identification, exercise identification, associated knowledge component identification, and transaction success.*

index	learner_id	skill_id	exercise_id	correct
0	0	1	1	0
1	0	1	1	1
2	0	1	2	1
3	0	3	4	0
4	0	2	3	1
5	1	1	1	1
6	1	2	3	1

Table 5.1: Example of a dataset that can be extracted from learner traces. Here, it concerns two learners: the first has done 5 transactions on 4 exercises related to 3 KCs; the second has done 2 transactions on 2 exercises related to 2 KCs. Indexing represents temporality.

Projection onto the model variables

First, for each learner, the data from the learner traces is projected on the variables of a dynamic Bayesian network with N_{traces} timeslices. Most of these variables are hidden. It is the case for all the transition and prerequisite auxiliary variables, which are intrinsically latent. It is also the case for the mastery variables of the KCs that are not assessed at each timeslice. The data on hidden variables is missing. This results in a massively incomplete dataset with a non-homogeneous number of columns. We call it the training extended dataset.

Example 5.2.2 (Training extended dataset)**.** *We return to Example 5.2.1 and focus on learner traces from the first learner. The learner model corresponding to this set of variables has 37 variables. The corresponding data entry has only five observed variables:* $X_1^0 = \neg x_1$, $X_1^1 = +x_1$, $X_1^2 = +x_1$, $X_3^3 = \neg x_3$ *and* $X_2^4 = +x_2$. *The other variables in the DBN are*

hidden. The corresponding entry in the training dataset projected on model variables is represented in Table 5.2.

learner	X_1^0	X_2^0	$Z_{3,1}^0$	$Z_{3,2}^0$	X_3^0	...	T_1^4	X_1^4	T_2^4	X_2^4	$Z_{3,1}^4$	$Z_{3,2}^4$	T_3^4	X_3^4
1	**0**	?	?	?	?	...	?	?	?	**1**	?	?	?	?

Table 5.2: Data entry corresponding to the learner traces of the first learner in the former dataset represented in Table 5.1. The "?" corresponds to hidden variables.

5.2.2 E-step: Completing the training extended dataset with inference

Similarly to the EM algorithm on a classic Bayesian network, the principle of the EM algorithm in a DBN is to complete each entry of the training extended dataset with inference. This is the E-step.

Classic EM algorithm approach

It is important to note that the whole DBN must be considered for inferring the state of hidden network variables. If a DBN can be resumed with two Bayesian networks, the Markov assumption that grants this simplification only holds thanks to the conditional independence of two non-successive timeslices given the full knowledge on a timeslice between the two. In our case, there is no timeslice with entire knowledge. Therefore, the entire DBN must be considered to infer the hidden variables' state correctly.

Let be a data entry $\mathbf{x} = (\mathbf{y}, \mathbf{z})$, with $\mathbf{y}$ the observed variables and $\mathbf{z}$ the hidden variables of the data entry. In practice, the E-step in the DBN of E-PRISM consists in computing the CPD $P(\mathbf{z} \mid \mathbf{y}, \theta)$ for each data entry $\mathbf{x}$ of the training extended dataset. Several approaches can be employed for inference, as detailed in Section 5.1.

Usually, in the EM algorithm, the probabilities associated with each configuration of the query variables $\mathbf{y}$ must be stored to perform the M-step. Nevertheless, we have seen that the data collected on the DBN representing the learner model suffer from an induced high scarcity. Consequently, the probability queries to compute at each E-step of the EM algorithm rely on many query variables, which imply two complications. On the one hand, for each data entry, the dimension of the state space of hidden variables is large, and this may induce memory issues with machines used for computation. During the learning process, the inference task is mobilized several times, and storing the distribution of hidden variables for each data entry is irrealistic. On the other hand, the number of probabilities representing the conditional probability distribution of interest is also very large. Before eventually storing the CPD that allows the completion of a data entry, one must be capable of computing it.

Stochastic approaches

We use stochastic variants of the EM algorithm, detailed in Section 3.2, to avoid storing the whole distribution for each entry in the data. The SEM and MCEM algorithms aim to sample the distribution rather than store it in the machine's memory during the learning process. Using these stochastic EM algorithm variants allows us to eliminate storing issues induced by the classic implementation of the EM algorithm.

The SEM algorithm in E-PRISM works the same way as the EM algorithm. Still, instead of computing and storing the joint posterior distribution for each data entry of the training extended dataset, one draws a sample from it. In the MCEM algorithm, we estimate the joint posterior distribution with multiple samples from it, leading to a Markov Chain Monte-Carlo (MCMC) approximation of the distribution. Its implementation is detailed in Algorithm 3.5.

Example 5.2.3 (Completed dataset)**.** *We suppose the data entry studied in Example 5.2.2. We assume this data entry is completed by sampling the joint posterior distribution with exact or approximate inference. The completed dataset is then represented in Table 5.3.*

learner	X_1^0	X_2^0	$Z_{3,1}^0$	$Z_{3,2}^0$	X_3^0	...	T_1^4	X_1^4	T_2^4	X_2^4	$Z_{3,1}^4$	$Z_{3,2}^4$	T_3^4	X_3^4
1	**0**	1	0	1	0	...	1	1	1	**1**	1	1	0	0

Table 5.3: Data entry of the completed dataset corresponding to the data entry reported in Table 5.2 in the SEM algorithm.

The unique difference between the SEM and the MCEM algorithm is the number of samples considered to approximate the joint posterior distribution during each E-step. The MCEM algorithm supposes an MCMC method that approximates the distribution. Instead of considering a unique sample for the distribution, n samples generated from the MCMC are considered. We represent in Table 5.4 how the data entry from the projected training dataset shown in Table 5.2 is processed during the E-step of the MCEM algorithm, with $n = 5$.

learner	X_1^0	X_2^0	$Z_{3,1}^0$	$Z_{3,2}^0$	X_3^0	...	T_1^4	X_1^4	T_2^4	X_2^4	$Z_{3,1}^4$	$Z_{3,2}^4$	T_3^4	X_3^4
1	**0**	0	0	1	0	...	1	1	1	**1**	1	1	0	0
1	**0**	1	0	1	1	...	1	1	1	**1**	1	1	1	1
1	**0**	0	0	1	0	...	1	0	1	**1**	0	1	1	0
1	**0**	0	0	1	0	...	1	1	1	**1**	0	1	0	0
1	**0**	1	0	1	0	...	1	1	1	**1**	0	1	1	0

Table 5.4: Data entries of the completed dataset corresponding to the data entry reported in Table 5.2 in the MCEM algorithm.

Inference techniques for completing the dataset

We keep using exact inference to sample the joint posterior distribution in the E-step, as it guarantees the convergence of the EM algorithm. However, the computations in exact inference are only tractable with constraints on the dimension of the DBN. Therefore, the KCs and transactions taken into account in the former training dataset must be limited. For instance, with a single core of Apple's M1 chip (3.2GHz with 16Go RAM), we saw in Section 5.1 that we can compute exact inference for data entries containing at most six transactions on a domain model composed of three KCs in a reasonable running time. Nevertheless, parameter learning requires inferring several complex distributions, so the limitations induced by exact inference come even quicker. Thus, the data entries of the learning dataset should rely on small numbers of transactions from learners to keep the exact inference usable for parameter learning. We understand that these constraints on the task of parameter learning are troublesome. Model learning is more efficient when there is a high quantity of provided data. Using exact inference in parameter learning for the learner model of the E-PRISM framework becomes a real issue.

Because exact inference is rapidly not tractable when the size of the DBN is too high, we use approximate inference to compute the joint posterior distribution relative to each data entry of the training extended dataset. As detailed in Section 5.1.4, Blocking Gibbs Sampling (BGS) is a relevant technique to greatly approximate distributions in the DBN of the E-PRISM learner model. BGS generates samples that approximate the distribution. For each training extended dataset data entry, instead of sampling from the exact inference, we consider the samples generated with BGS. Because we showed that the estimator of the Blocking Gibbs sampling effectively converges to the distribution obtained from exact inference, the MCEM algorithm should converge as well as the classical EM algorithm or the stochastic EM algorithm.

5.2.3 M-step: Updating the model parameters from the completed dataset

After the E-step, we have a completed dataset that is composed of an entry per learner, containing every variable of the DBN that models the knowledge state of the learner during all its transactions. The DBN supposes the stationarity of the modeled process. This means that the parameters of the model are time-independent. Then, we must return to a dataset that does not consider differences between timeslices to perform the M-step. We recall the M-step aims at computing updated parameters that optimize the model with the completed dataset.

Decomposing the completed dataset into the initial and transition datasets

More precisely, a DBN is a set of two Bayesian networks: it then requires two datasets from which parameters are updated. On the one hand, the dataset $\mathcal{D}_0$ is the dataset that resumes the knowledge states at initial timeslice for every learner. This implies that the dataset $\mathcal{D}_0$

is composed of one row per learner in the former dataset. On the other hand, the dataset $\mathcal{D}_{\rightarrow}$ is the dataset that resumes the transition between two timeslices $t-1$ and t for every $t > 0$. A learner that would have done N_{traces} transactions would then result in N_{traces} rows in the dataset $\mathcal{D}_{\rightarrow}$.

Example 5.2.4 (Initial and transition datasets)**.** *We describe the initial and transition datasets $\mathcal{D}_0$ and $\mathcal{D}_{\rightarrow}$ generated from Example 5.2.3. We represent in Tables 5.5 (resp. in Table 5.6) the data entries in $\mathcal{D}_0$ (resp. in $\mathcal{D}_{\rightarrow}$) induced by the data entry of the completed dataset represented in Table 5.3.*

learner	X_1^0	X_2^0	$Z_{3,1}^0$	$Z_{3,2}^0$	X_3^0
1	**0**	1	0	1	0

Table 5.5: Data entry in $\mathcal{D}_0$ corresponding to the data entry reported in Table 5.3.

learner	X_1^{t-1}	X_2^{t-1}	X_3^{t-1}	T_1^t	X_1^t	T_2^t	X_2^t	$Z_{3,1}^t$	$Z_{3,2}^t$	T_3^t	X_3^t
1	**0**	1	0	1	**1**	1	1	0	1	0	0
1	**1**	1	0	1	**1**	1	1	0	1	0	0
1	**1**	1	0	1	1	1	1	0	1	0	**0**
1	1	1	**0**	1	1	1	**1**	1	1	0	0

Table 5.6: Data entries in $\mathcal{D}_{\rightarrow}$ corresponding to the data entry reported in Table 5.3.

Computing parameters from the initial and transition datasets

The M-step of the EM algorithm consists in maximizing the likelihood of the model in regard to these two datasets. For a Bayesian network, it is simply finding the parameters that optimize the likelihood. The BN parameters that maximize the likelihood given a complete dataset are defined in Equation 3.28. Because E-PRISM relies on two BNs, this task must be done twice. $\mathcal{D}_0$ is used to compute the parameters of $\mathcal{B}_0$ and $\mathcal{D}_{\rightarrow}$ is used to compute the parameters of $\mathcal{B}_{\rightarrow}$. We represent the updating formulas for $\mathcal{B}_0$ parameters in Table 5.7, and those for $\mathcal{B}_{\rightarrow}$ parameters in Table 5.8.

In a nutshell, in order to perform the EM algorithm in a DBN, one must keep in mind that the E-step must be performed on the unrolled DBN, as the incompleteness of the dataset makes the conditional independence between variables no longer truthful. However, once the dataset is completed, the M-step is performed on datasets assuming the stationarity of the process stated by the model parameters.

parameter	updating formula from $\mathcal{D}_0$
$p_{\mathfrak{X}}$	$\frac{\#(X^0 = +x)}{\#\left((X^0 = \neg x) \cup (X^0 = +x)\right)}$
$q^0_{\mathfrak{X},i}$	$1 - \frac{\#\left((Z^0_{\mathfrak{X},i} = +z_i) \cap (Pa^0_{\mathfrak{X},i} = +pa_i)\right)}{\#\left(Pa^0_{\mathfrak{X},i} = +pa_i\right)}$
$s^0_{\mathfrak{X},i}$	$\frac{\#\left((Z^0_{\mathfrak{X},i} = +z_i) \cap (Pa^0_{\mathfrak{X},i} = \neg pa_i)\right)}{\#\left(Pa^0_{\mathfrak{X},i} = \neg pa_i\right)}$

Table 5.7: Table of the updating formulas from $\mathcal{D}_0$ for the parameters of the BN $\mathcal{B}_0$ relative to a knowledge component $\mathfrak{X}$.

parameter	updating formula from $\mathcal{D}_{\rightarrow}$
$l_{\mathfrak{X}}$	$l_{\mathfrak{X}} = \frac{\#\left((T^t_{\mathfrak{X}} = +t) \cap (X^t = \neg x)\right)}{\#\left(X^t = \neg x\right)}$
$f_{\mathfrak{X}}$	$f_{\mathfrak{X}} = \frac{\#\left((T^t_{\mathfrak{X}} = \neg t) \cap (X^t = +x)\right)}{\#\left(X^t = +x\right)}$
$q_{\mathfrak{X},i}$	$1 - \frac{\#\left((Z^t_{\mathfrak{X},i} = +z_i) \cap (Pa^t_{\mathfrak{X},i} = +pa_i)\right)}{\#\left(Pa^t_{\mathfrak{X},i} = +pa_i\right)}$
$s_{\mathfrak{X},i}$	$\frac{\#\left((Z^t_{\mathfrak{X},i} = +z_i) \cap (Pa^t_{\mathfrak{X},i} = \neg pa_i)\right)}{\#\left(Pa^t_{\mathfrak{X},i} = \neg pa_i\right)}$

Table 5.8: Table of the updating formulas from $\mathcal{D}_{\rightarrow}$ for the parameters of the BN $\mathcal{B}_{\rightarrow}$ relative to a knowledge component $\mathfrak{X}$.

5.2.4 Study of the MCEM algorithm for parameter learning in E-PRISM

Now the theoretical background of parameter learning is elicited, we study the convergence of the presented techniques for learning the parameters of the E-PRISM learner model from synthetic data generated from E-PRISM itself.

First, we generate synthetic datasets with missing values from the E-PRISM framework to evaluate the performance of parameter learning from them. We study the convergence of the parameter learning task thanks to the Negative Log-Likelihood (NLL). The NLL is supposed to decrease to zero as much as the learned model fits the dataset. We also measure the effectiveness of the parameter learning task by computing the Kullback-Leibler (KL) divergence between the learned BN distribution and the target BN distribution for different values of the model hyperparameters.

Generating a synthetic dataset with missing values

First, we define the E-PRISM learner model that is used for generating synthetic data. We suppose a domain model with two KCs $\mathfrak{A}$ and $\mathfrak{B}$. It assumes the existence of the prerequisite relationship $\mathfrak{A} \rightarrow \mathfrak{B}$. We set up the parameters of the learner model as follows:

- KC parameters:
  - "prior" parameters: $p_{\mathfrak{A}} = 0.1$ and $p_{\mathfrak{B}} = 0.08$
  - "learn" parameters: $l_{\mathfrak{A}} = 0.3$ and $l_{\mathfrak{B}} = 0.2$
  - "forget" parameters: $f_{\mathfrak{A}} = 0.01$ and $f_{\mathfrak{B}} = 0.04$
- Prerequisite parameters
  - q parameter: $q_{\mathfrak{B},\mathfrak{A}} = 0.1$
  - s parameter: $s_{\mathfrak{B},\mathfrak{A}} = 0.05$

These parameter values have been chosen arbitrarily. Still, they aim to correspond to credible values. The KC parameters are chosen accordingly with plausible values [dBCA08]. The prerequisite parameters correspond to a strong prerequisite relationship between $\mathfrak{A}$ and $\mathfrak{B}$.

We generate a complete dataset representing all DBN variables of the E-PRISM learner model for 200 learners that would have realized 3 exercises each. Each entry in the complete dataset consists of 13 columns. These values have been chosen because the synthetic dataset should be used to compare exact and approximate inference tasks in parameter learning. Thus, it must induce tractable computations for exact inference. Afterward, we extract a dataset with missing data from the complete dataset. We suppose that only one mastery variable is known at each timestamp and that every auxiliary variable is unknown. We obtain a dataset composed of entries in the shape of the one in Table 5.2.

Metrics to measure the convergence of parameter learning

We learn the parameters of a newly-instantiated E-PRISM learner model from these synthetic data. We suppose the learner model to relates to the same domain model as the one used for generating the training dataset. The parameters of the learner model are randomly generated. We apply two parameter learning algorithms to the synthetic data: MCEM with exact inference and MCEM with BGS approximate inference.

First, we compare the learner model with learned parameters with training incomplete data to measure the convergence of the parameter learning task. To do so, we study the value of the Negative Log-Likelihood (NLL) of the training data with the learned model. The NLL is already used in the EM algorithm as a criterion to assert convergence. The final NLL value is obtained at the last step of the learning process and is supposed to be the minimum value. The NLL provides a measure of convergence in the learning process. In addition, it

can also be used to compare the performance of different parameter learning algorithms on the same dataset. A lower NLL indicates a better fit of the model to the data, so algorithms that produce a lower NLL are considered more accurate and effective at estimating the model parameters. Because the EM algorithm only assures convergence to local minima, the comparison of the NLL values can be employed to compare the performance of different parameter learning procedures of the same algorithm on the same dataset. A lower NLL indicates that learned model parameters better correspond to the data.

Second, we compare the learned model with the one used for generating the data. Because the data is generated from instances of the E-PRISM learner model, we can directly compare the DBN that generated the data with the DBN obtained from the parameter learning procedure. We use Kullback-Leibler (KL) divergence to compare them. KL divergence is employed to compare the convergence of different parameter learning procedures in BNs. It measures the difference between the distribution of the estimated model parameters and the true underlying distribution of the parameters. Suppose one parameter learning procedure can estimate the parameters of the Bayesian network more accurately than another. In that case, the KL divergence between the estimated and true distributions of the parameters will be smaller for the more accurate procedure. We will study the evolution of the KL divergence as a function of the hyperparameters of the parameter learning algorithms.

We want to demonstrate the capacity of the E-PRISM parameter learning procedure to converge. Therefore, the parameter learning algorithm is applied to the whole synthetic dataset. The purpose of the E-PRISM framework is to be directly used in an ITS. Therefore, we evaluate the capacity of the learning algorithm to learn from an entire dataset that would be produced on the ITS, and we study how the learning procedure converges to the optimal fit of the whole data. The performance of the learning algorithm when evaluated on new data will be studied in Chapter 6.

Hyperparameters of the learning task

The learning task relies on several hyperparameters that must be defined beforehand. Hyperparameters are values that control the learning process itself rather than the model's behavior. Hyperparameters are often set by the practitioner and can significantly impact the performance of the model. As such, it is essential to select the hyperparameters to achieve optimal results carefully.

On the one hand, hyperparameters are intrinsically associated with the chosen parameter learning algorithm. These hyperparameters directly depend on the underlying inference algorithm. For instance, the MCEM algorithm with exact inference relies on the number of samples taken into account for each data entry N_{sample}. In contrast, the MCEM algorithm with BGS approximated inference depends on the number of Gibbs iterations N_{Gibbs} and the burn-in period M. We saw in Section 5.1.3 that the burn-in period doesn't impact the convergence of the inference task. Therefore it should not impact the convergence of

parameter learning, and it will not be studied here.

On the other hand, because the family of EM algorithms only assures convergence to a local minimum, we have decided to perform multiple parallel runs of the learning procedure potentially. This introduces a new hyperparameter N_{EM}, the number of parameter learning procedures done in parallel. It avoids being trapped in local equilibria. It may even be mandatory to consider several procedures simultaneously, as there may be a high variance between two similar procedures whatever the chosen algorithm for parameter learning,

Study of the inference hyperparameters

First, we focus on the inference hyperparameters, namely the number of samples N_{samples} for the MCEM learning algorithm with exact inference and the number of Gibbs iterations N_{Gibbs} for the MCEM learning algorithm with BGS approximated inference. We study the effect of these hyperparameters on the learning procedure from the synthetic dataset. We investigate the differences in the computational time taken for parameter learning. Then, we continue analyzing the results given by the NLL and KL divergence values. We represent in Figure 5.13 the running time of the parameter learning task from the synthetic dataset as a function of N_{samples} and N_{Gibbs}.

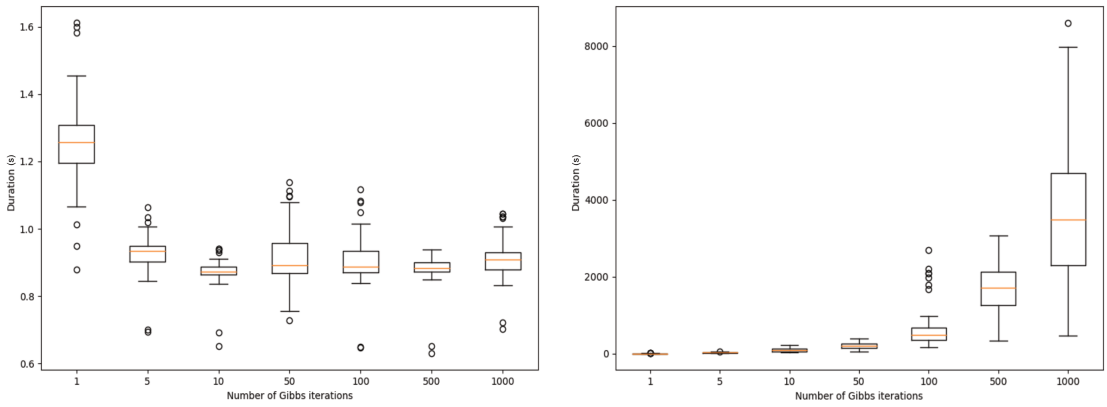


Figure 5.13: On the left: running time of the MCEM algorithm with exact inference as a function of N_{samples}. On the right: running time of the MCEM algorithm with approximate inference task as a function of N_{Gibbs}. Parameter learning is done from the dataset generated on the 2 KCs learner model.

We can figure that the running time of the learning procedure with the MCEM learning algorithm with exact inference doesn't depend on N_{samples}. Indeed, we observe that every running time stays the same, apart from the running time when $N_{\text{samples}} = 1$, which is slightly longer. On the contrary, the running time of the learning procedure with the MCEM algorithm with BGS approximated inference grows exponentially with N_{Gibbs}. This result is entirely logical according to the results observed on E-PRISM inference. On the one hand, MCEM with exact inference only computes the posterior distribution once, no matter the value of N_{samples}, and then it samples it N_{samples} times. We guess that the exception of

$N_{\text{samples}} = 1$ is uniquely due to extra list manipulation in `Python`.

On the other hand, MCEM with approximated inference relies on the Blocking Gibbs sampling inference, whose running time linearly grows with N_{Gibbs}. Because MCEM mobilizes inference for each learner entry of the dataset, the computational complexity of the parameter learning algorithm is an exponential function of N_{Gibbs}. We already saw in Section 5.1 that exact inference is not tractable for large networks. Even if the MCEM algorithm with exact inference leads to quicker results on a 2 KCs domain model, we can show that using the MCEM algorithm with exact inference is rapidly intractable for larger domain models. On the contrary, the MCEM algorithm with BGS approximate inference stays tractable for low values of N_{Gibbs}, no matter the size of the domain model.

We now represent the NLL of the training data and the KL divergence between target and learned distributions as functions of the value of N_{samples} for the MCEM learning algorithm with exact inference in Figure 5.14.

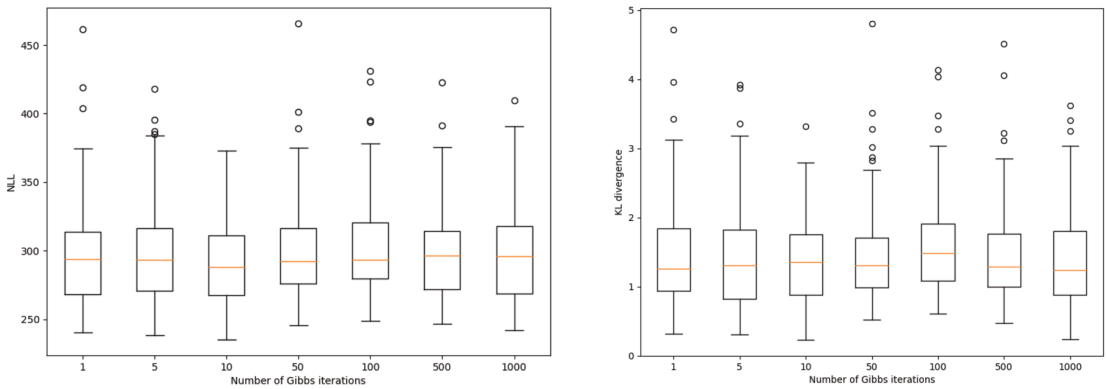


Figure 5.14: On the left: NLL of the training data as a function of the value of N_{samples} for the MCEM learning algorithm with exact inference. On the right: KL divergence between target and learned distributions as a function of the value of N_{samples} for the MCEM learning algorithm with exact inference.

We can observe in Figure 5.14 that both the NLL and the KL divergence stay the same, whatever the value of N_{samples}. The distribution from which the completed dataset is sampled is computed exactly. It then directly gives coherent samples even if only one sample is considered. Then, because there is a sufficient number of entries in the dataset, the stochastically approximated distribution with exact inference sampling is close enough to the target distribution.

We represent the NLL of the training data and the KL divergence between target and learned distributions as functions of the value of N_{Gibbs} for the MCEM learning algorithm with BGS approximated inference in Figure 5.15.

Contrary to the MCEM algorithm with exact inference, we observe in Figure 5.15 that the value of N_{Gibbs} impacts both the NLL and the KL divergence. Indeed, we can see that their value exponentially decreases with the value of N_{Gibbs}. As reported in Section 5.1,

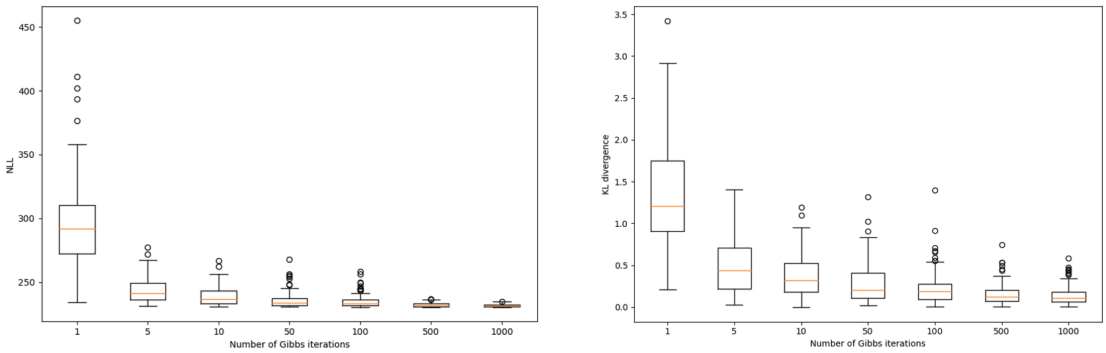


Figure 5.15: On the left: NLL of the training data as a function of the value of N_{Gibbs} for the MCEM learning algorithm with BGS approximated inference. On the right: KL divergence between target and learned distributions as a function of the value of N_{Gibbs} for the MCEM learning algorithm with BGS approximated inference.

the higher the number of Gibbs iterations, the better the approximation of the inference task. Consequently, multiple Gibbs iterations are required to correctly sample the target distribution and complete the dataset with missing data.

We can extract a critical N_{Gibbs} value from which the NLL is low enough for the parameter learning procedure to be close enough to correct parameter distribution. We denote it $N_{\text{Gibbs},c}$. We compare the NLL value given by the MCEM algorithm with approximated inference with the one obtained from MCEM with exact inference to determine this value. We observe that the NLL and the KL divergence obtained with approximate inference are lower than those obtained with exact inference from $N_{\text{Gibbs}} > 1$. We also study the disintegration constant that appears when considering the evolution of NLL and KL divergence as exponential decay. We observe their disintegration constant looks similar. We approximate the critical value by $5 < N_{\text{Gibbs},c} < 10$. by studying the tangent of the exponential decay in Figure 5.15.

Study of the parameter learning hyperparameter

We study the influence of the parameter learning hyperparameter, namely the number of simultaneous instances of the EM algorithm N_{EM}. This hyperparameter must be considered for both the MCEM algorithm with exact inference and the MCEM algorithm with BGS approximated inference. We suppose that the hyperparameters N_{samples} and N_{Gibbs} are fixed to the following value: $N_{\text{samples}} = N_{\text{Gibbs}} = 10$.

We represent the KL divergence as a function of the number of EM algorithm instances in Figure 5.16. We represent the distribution of 10 realizations of N_{EM} simultaneous instances of the EM algorithm with boxplots. For each of the 10 realizations, we select the best set of learned parameters among the results of the N_{EM} EM algorithm instances according to their value of NLL.

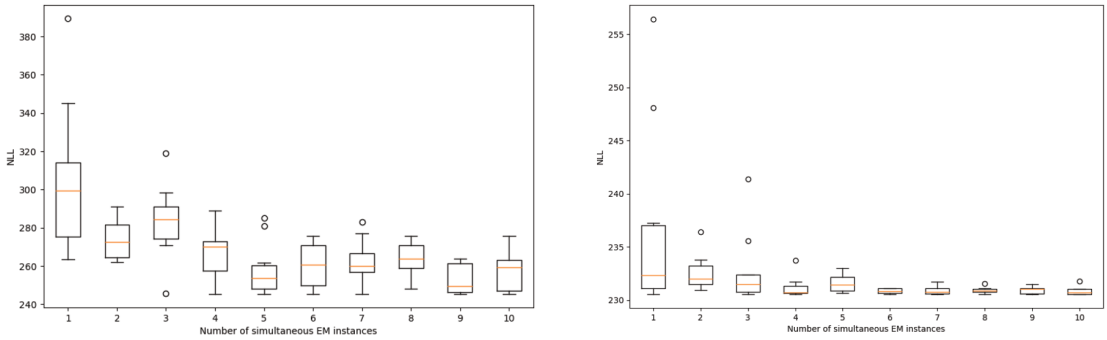


Figure 5.16: On the left: distribution of the NLL associated with the best set of parameters among N_{EM} instances of the MCEM algorithm with exact inference. On the right: distribution of the NLL associated with the best set of parameters among N_{EM} instances of the MCEM algorithm with BGS approximated inference.

Figure 5.16 shows that the results of the parameter learning algorithm are subject to high variance. We should consider the best result from multiple instances of the EM algorithm rather than a unique instance to reduce the risk of considering not optimal learned parameters. We observe the best NLL is stable when considering at least 5 simultaneous instances for the MCEM algorithm with exact inference. Regarding the MCEM algorithm with approximated inference, we can spot extreme values for runs with $N_{\text{EM}} < 5$. The stability of the results also comes from 5 simultaneous instances as well.

Study of the dataset hyperparameters

Finally, we focus on the dataset hyperparameters. Dataset hyperparameters are values that describe the specifications of the training dataset. In our case, the unique dataset hyperparameter is the maximum number of transactions per learner. As reported in Section 5.1, we saw that the number of transactions that can be handled is a considerable limitation of both exact and approximated inference. Inference is mobilized multiple times during the parameter learning procedure, so we must study the effect of the number of learner transactions on the running time of the parameter learning algorithm. We focus on the MCEM algorithm with BGS approximate inference, and we set $N_{\text{Gibbs}} = 10$.

We suppose a new synthetic dataset relying on the same domain model as before. We suppose 12 transactions per learner. We iteratively consider 2 to 12 transactions per learner for the parameter learning procedure. We suppose the other hyperparameters to be set to represent the running time of the parameter learning procedure as a function of the number of learner transactions per learner in Figure 5.17.

We can observe that the running time rapidly skyrockets. To keep the computation tractable when the parameter learning task is done multiple times in a bigger procedure, we recommend the maximum number of transactions per learner to be lower than 10 transactions, preferably

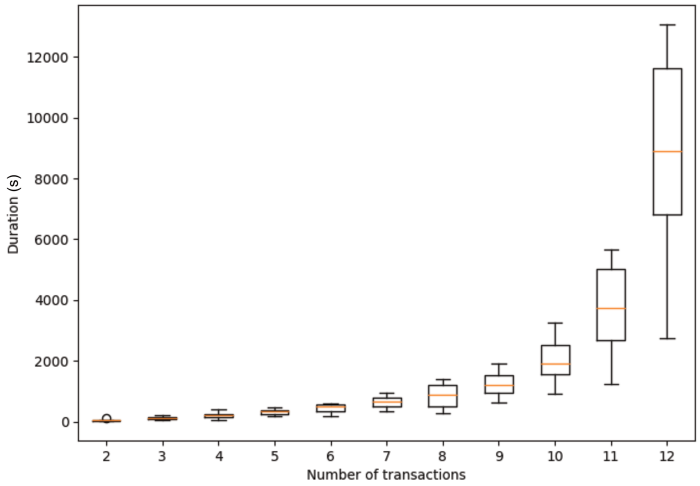


Figure 5.17: Running time of the parameter learning task as a function of the number of transactions taken into account in the training dataset.

even lower than 7 transactions.

5.3 Discussion

In this chapter, we have studied the required techniques to use the E-PRISM framework for student modeling. They provide tools to ITS for diagnosing and predicting learners' knowledge states over time and updating the learner model. These have been implemented in the E-PRISM `Python` package.

First, we have exposed the techniques for diagnosing learners' knowledge states over time, thanks to inference techniques in Bayesian networks. We have seen that usual inference techniques in Bayesian networks suffer from intractability and convergence issues. Our findings demonstrate that traditional exact inference is hindered by its rapid intractability in the context of E-PRISM, and Gibbs sampling fails to converge due to the reducibility of the Markovian process caused by the ICI-model CPDs. This is why we proposed a novel approach based on Blocking Gibbs Sampling (BGS) for approximate inference in ICI-based Bayesian networks. Our experiments showed that this approach allows for more tractable convergence than exact inference and provides accurate results.

Then, we introduced the procedure for updating the learner model from data. Specifically, we presented parameter learning in the E-PRISM learner model. We showed how the interpretability of the learned parameters could provide insights into the prerequisite structure of the domain knowledge and insights into the learning and forgetting phenomena.

These contributions provide a solid foundation for applying E-PRISM in ITS, where it can help optimize the delivery of personalized instruction by diagnosing the learners' knowledge state and predicting their performance. We will study the capacity of our framework to predict learners' performance and compare it with other student modeling techniques in Chapter 6. In particular, we will wonder how the learner model updating procedure can produce insights into the prerequisite structure of the domain model.

Chapter 6

Applications of the E-PRISM framework

We have presented the theory behind the elements of the E-PRISM framework in Chapters 4 and 5. In this chapter, the objective is to demonstrate the E-PRISM capacity to be deployed to model the mastery state of Knowledge Components (KCs) and, therefore, to accurately predict the learners' performance on an Intelligent Tutoring Systems (ITS). In particular, we examine the capacity of our model to identify prerequisite relationships between the KCs of the domain model.

To evaluate the capacity of the model to predict mastery states and learner performance, we apply it to datasets of learner transactions related to various learning environments. We compare the E-PRISM learner model's predictions to the actual learner performance on different data sources and evaluate the accuracy of these predictions using insightful metrics.

In addition to its potential for predicting learner performance and mastery states, we examine the potential of the E-PRISM framework for identifying the prerequisite structure of the domain model. Rather than being determined through explicit patterns in the data, these relationships are revealed through the learned values of the model parameters. We use the parameter learning procedure of E-PRISM to discover KC prerequisite relationships.

Overall, the results of our analysis demonstrate the capacity of the model to accurately predict learner performance and mastery states, as well as its potential for identifying prerequisite relationships and providing an adaptive domain model. The following sections describe our methodology and present the results of the evaluation carried out in detail.

6.1 Study of the E-PRISM prediction performance

In this section, we describe the methodology we used to evaluate the capacity of the E-PRISM learner model to predict the learner's performance on an ITS. Accurate prediction of learner performance is essential for an ITS to adapt to individual learners' needs and abilities and provide personalized feedback and support.

As reported in Chapter 5, the running time of the parameter learning task for E-PRISM, in our execution context, is very high when considering more than two KCs in the domain model. To overcome this and provide a glimpse at comparing E-PRISM in predicting performance with other cognitive diagnosis algorithms, we study knowledge components by pairs only.

6.1.1 Measuring the performance in predicting learner's performance

First, we differentiate two uses of the word "performance" to understand the following better. In this study, the performance of algorithms refers to the operational characteristics of the algorithms, such as their accuracy and efficiency. In contrast, learners' performance refers to the outcomes or results of their interactions with the ITS. In our context, these outcomes correspond to the correctness of their transactions.

State-of-the-art algorithms for learner's performance prediction

In this study, we compare the performance of several algorithms specifically used for predicting the learners' answers from data. These algorithms have been presented in Section 2.2. We have chosen to compare the following algorithms:

- Logistic regression algorithms: IRT, PFA, DAS3H, and BestLR;
- Bayesian knowledge tracing: classic BKT and an extended version (when possible), which considers a "forget" parameter, item-dependent slip and guess parameters, and KC-dependent learn parameters;
- E-PRISM: E-PRISM learner model trained with MCEM with exact inference (when possible) and with MCEM with BGS approximate inference – see Section 5.2 for more details.

We recall that these algorithms use statistical and machine learning techniques to analyze student data and predict how students will perform on future assessments. By evaluating the performance of these different algorithms, we can gain insight into which approaches are the most effective at supporting personalized and adaptive instruction in ITS.

The BGS approximate inference uses the values determined in Section 5.2 to guarantee the convergence of the parameter learning task. The number of Gibbs iterations is set to $N_{\text{Gibbs}} = 10$, and the burn-in period is $M = 0$. The exact inference is also performed with a number of samples $N_{\text{samples}} = 10$ to simplify the comparison between the two approaches.

Discussion on available metrics for learner's performance prediction

Several metrics can be used to measure the capacity of a model to predict the learner's performance from learner traces on an ITS. Among the standard metrics, we observe in the literature that the main employed ones include the following:

- The Accuracy (ACC), which measures the percentage of correct predictions made by the algorithm;
- The Mean Absolute Error (MAE), which measures the average prediction error;
- The Root Mean Square Error (RMSE), which measures the difference between the predicted and actual values;
- The Area Under the ROC Curve (AUC), which measures the ability of the algorithm to distinguish between positive and negative cases.

Pelánek differentiates "error" metrics, such as mean-error or RMSE, for which a lower value is better, and "rewards" metrics, such as ACC and AUC, for which a higher value is better [Pel15]. He also categorizes these metrics into three categories.

The first category of metrics consists of a probabilistic understanding of predictions and errors. It comprises MAE, RMSE, and every likelihood-based metric. Such metrics report the difference between probabilistic predictions and actual results. Pelánek recommends using these metrics for predictions of learner performance. The second type of metric consists of a qualitative understanding of the errors. It only focuses on whether the classification prediction is correct, no matter the predicted probability value. Finally, the Receiver Operating Characteristics (ROC) curve and the related AUC metric emphasize the third category of metrics. These metrics compare the results obtained by the prediction tasks relatively with each other. For instance, the AUC score equals "the probability that a randomly selected positive observation has a higher predicted score than a randomly selected negative observation."

We then understand that it is crucial to consider which metric is the most relevant in context. The specific goals and objectives of the predictive modeling task give insights into the metrics to use. Pelánek recommends not to use MAE at all, as it is described as a misleading metric [Pel15]. Moreover, he suggests preferring RMSE or likelihood-related metric to AUC for model comparison [Pel15]. The AUC metric can be used in addition to either RMSE or likelihood-related metric to provide a more interpretable understanding of the results. Pardos and Yudelson even assert that "AUC should not be used to determine the relative goodness of models based on prediction performance if the underlying goal is to rank models based on knowledge estimation goodness." [PY13]

Use of cross-validation

Cross-validation is a resampling procedure used to evaluate the performance of a machine-learning model. It is a way of evaluating the model on new data that was not used to train it. In the context of our research work, we use holdout cross-validation.

To do so, we split the dataset into training and validation sets using the 80/20 proportion. Parameter learning is performed on the training set, using classic 5-fold cross-validation, the most common type of cross-validation. In k-fold cross-validation, the training dataset is divided into k folds. The model is then trained and evaluated k times, using a different fold as the test set each time. The training performance metric is then averaged across the k iterations. It provides a more robust estimate of the model's performance based on multiple train and test splits rather than just one.

Then, it is evaluated on the validation set that has not been "seen" by the model yet. Cross-validation aims to estimate the model's performance on new data. One of the main benefits of cross-validation is that it allows the model to be trained and evaluated on different data subsets, which helps reduce the risk of overfitting. In summary, cross-validation is a valuable technique for evaluating the performance of a machine learning model, as it helps ensure that the model will generalize to new, unseen data and for identifying any overfitting that may have occurred during training.

Cluster computing

Using cross-validation and more extended datasets complexifies the computations required in this chapter. While Apple's M1 chip was used in the computations in Chapter 5 to perform parameter learning, we accessed clustered computing machines to perform efficient parallel computations with the E-PRISM framework for the experiments of this chapter. Inference and parameter learning tasks have been dispatched among the threads of the cluster computing nodes. The nodes are described in Table 6.1.

Processors	RAM memory	Threads
2 x Intel Xeon X5677	144 Go	16 threads @ 3.47 GHz
2 x Intel Xeon X5690	144 Go	24 threads @ 3.47 GHz
2 x Intel Xeon E5-2690v3	192 Go	48 threads @ 2.60 GHz
2 x Intel Xeon X5570	144 Go	16 threads @ 2.93 GHz
2 x Intel Xeon E5-2650v3	256 Go	40 threads @ 2.30 GHz

Table 6.1: Specifications of the machine in the cluster.

Using clustered computing machines allowed us to parallelize the computations with more simultaneous processes. However, the parameter learning task cannot be dispatched on

multiple threads because the algorithm is iterative. The RAM memory attached to each thread is the total RAM of the machine divided by the number of total threads. Available RAM is then similar to Apple's M1 chip. This is the limiting factor in the complexity of our framework.

6.1.2 Predicting the learners' performance from synthetic data

First, we study the performance of the different predicting algorithms to predict the learner's performance from synthetic data generated from E-PRISM. We want to compare the capacity of E-PRISM to predict learner performance with the capacity of the other main cognitive diagnosis algorithms when the data is well-tailored to our custom model. For each of the studied algorithms, we look at the RMSE and AUC values obtained from training with the 5-fold cross-validation with the 80/20 proportion and the evaluation of the validation set of the synthetic dataset.

Synthetic dataset and E-PRISM settings

The synthetic dataset is generated from E-PRISM. We suppose the domain model consists of two knowledge components $\mathfrak{A}$ and $\mathfrak{B}$ related to each other with the prerequisite relationship $\mathfrak{A} \to \mathfrak{B}$. We instantiate a learner model associated with this domain model. We set the parameters to the following values:

- "prior" parameters: $p_{\mathfrak{A}} = 0.1$ and $p_{\mathfrak{B}} = 0.08$
- "learn" parameters: $l_{\mathfrak{A}} = 0.3$ and $l_{\mathfrak{B}} = 0.2$
- "forget" parameters: $f_{\mathfrak{A}} = 0.01$ and $f_{\mathfrak{B}} = 0.04$
- q parameter: $q_{\mathfrak{B},\mathfrak{A}} = 0.1$
- s parameter: $s_{\mathfrak{B},\mathfrak{A}} = 0.05$.

As stated in Section 5.2, these parameter values used for data generation are arbitrarily chosen, but they correspond to plausible values for a strong prerequisite relationship [dBCA08]. We generate a complete dataset from the learner model with 200 learners and 7 transactions per learner. Then, we extract a dataset with missing data from the complete dataset.

One of the main objectives of the following experiments is to study the impact of the prerequisite relationships on the learner's performance prediction. To this end, we introduce three E-PRISM learner models ($e\Delta$). We compare the performance of $e\Delta_{\varnothing}$, $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$, and $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$, namely the E-PRISM learner models considering the possible configurations for the prerequisite relationship (respectively $\varnothing$, $\mathfrak{A} \to \mathfrak{B}$ and $\mathfrak{B} \to \mathfrak{A}$). We investigate if we can notice an impact of the prerequisite relationship on the performance of the E-PRISM learner model. We also compare the performance of $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$, which considers the correct prerequisite structure, with the state-of-the-art learner's performance prediction algorithms.

Synthetic data restricted to 3 transactions per learner

We have seen in Section 5.2 that parameter learning for E-PRISM can be done in two different ways. On the one hand, the MCEM algorithm with exact inference is rapidly limited when considering large numbers of transactions for each learner. On the other hand, the MCEM algorithm with approximate inference works for higher values of the number of learner's transactions.

As a first step, we consider the synthetic dataset restricted to 3 transactions per learner to perform E-PRISM parameter learning with exact inference. The purpose of this tiny dataset is to study the impact of the inference technique employed during MCEM parameter learning in E-PRISM. For each predicting algorithm, we report in Table 6.2 the mean and the best RMSE and AUC values obtained during cross-validated training, and the RMSE and AUC values obtained on the validation set from the model learned on the training set.

Algorithm	Mean train RMSE	Best train RMSE	Mean train AUC	Best train AUC	Validation RMSE	Validation AUC
IRT	0.395	0.344	0.631	0.675	0.428	0.659
PFA	0.374	0.361	0.706	0.773	0.393	0.744
DAS3H	0.369	0.315	0.699	0.778	0.383	0.758
Best-LR	0.354	0.314	0.757	0.835	0.364	0.805
Classic BKT	0.369	0.288	0.727	0.791	0.387	0.758
Extended BKT	0.371	0.358	0.727	0.849	0.386	0.758
Exact $e\Delta_{\varnothing}$	0.349	0.319	0.782	0.839	0.390	0.727
Exact $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.335	0.279	0.823	0.873	0.369	0.800
Exact $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.372	0.309	0.681	0.775	0.395	0.764
Approx. $e\Delta_{\varnothing}$	0.349	0.307	0.775	0.827	0.389	0.727
Approx. $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.336	0.278	0.820	0.866	0.374	0.791
Approx. $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.377	0.319	0.694	0.773	0.400	0.748

Table 6.2: RMSE and AUC values obtained from learner's performance prediction algorithms on synthetic data. The synthetic dataset consists of 3 transactions per learner, decomposed into training and validation sets. Parameter learning is done with cross-validation on the training set, and the algorithms' performance is evaluated on the validation set.

Contrary to the concerns of Pelánek [Pel15], we can remark in Table 6.2 that RMSE and AUC values are coherent with each other and lead to the same results on synthetic data. This report encourages us to only focus on studying RMSE values.

Concerning algorithms' performance, we observe exact and approximate alternatives of the E-PRISM parameter learning to give similar results. According to the E-PRISM parameter learning procedure study with both exact and approximate inference, similar performances are expected between exact and approximate $e\Delta$s. The difference between the two probably depends more on the variance due to the low number of EM initializations N_{EM} than on the type of inference technique. We also notice in Table 6.2 that the performance of exact and approximate $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ is better than the others. They perform best among the other $e\Delta$s (validation RMSE -0.021, validation AUC $+0.031$ for exact inference; validation RMSE -0.015, validation AUC $+0.043$ for approximate inference). Thus, the E-PRISM learner model that relies on the prerequisite structure of the domain model described in the synthetic dataset outperforms.

We can also observe that the logistic regression models exhibit a similar order to those reported in the literature [Pel17, CPBV19, GKS+20, SWHM22]. Best-LR is the most performant logistic regression algorithm, followed by DAS3H and PFA. IRT shows poor results compared to the others (validation RMSE $+0.035$, validation AUC -0.085 compared to the penultimate logistic regression algorithm regarding performance). More surprisingly, Best-LR is a more performant prediction algorithm than E-PRISM on the E-PRISM synthetic data. It outperforms $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$, which considers the correct prerequisite structure of the domain model, by a little (validation RMSE -0.004, validation AUC $+0.005$ for exact inference).

Finally, classic BKT has similar results with $e\Delta_{\varnothing}$. Their validation RMSE values are very close (0.387 for classic BKT and 0.390/0.389 for exact/approximate $e\Delta_{\varnothing}$). Also, the AUC value of classic BKT (0.743) is included in the interval between AUC values of exact (0.750) and approximate (0.736) $e\Delta_{\varnothing}$. This result sounds entirely logical, as the DBN used in E-PRISM when no prerequisite relationships are considered consists of parallel BKT models (without "slip" and "guess" parameters) without any interactions between each other.

Full synthetic dataset

We now train and evaluate each learner model on the full synthetic dataset by considering all 7 transactions per student. We have seen in Section 5.2 that the MCEM algorithm with approximate inference remains tractable when considering datasets with a number of transactions per learner of this order of magnitude. However, parameter learning with exact inference MCEM is not tractable for such a number of transactions. The parameter learning of E-PRISM is then studied with BGS approximate inference only. We represent the RMSE and AUC values obtained from training and evaluating in Table 6.3 for the 7-transactions synthetic dataset.

We notice that $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ is the most performant predicting algorithm on the 7-transactions synthetic data (validation RMSE -0.011, validation AUC $+0.020$ compared to Best-LR). The other $e\Delta$s lead to less notable performances, with $e\Delta_{\varnothing}$ still being more performant than $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$. We denote a large gap of performances between $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ and $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$ (RMSE

Algorithm	Mean train RMSE	Best train RMSE	Mean train AUC	Best train AUC	Validation RMSE	Validation AUC
IRT	0.462	0.450	0.688	0.729	0.484	0.649
PFA	0.384	0.371	0.845	0.870	0.409	0.814
DAS3H	0.383	0.361	0.844	0.870	0.408	0.814
Best LR	0.369	0.346	0.870	0.901	0.396	0.845
Classic BKT	0.381	0.366	0.847	0.863	0.408	0.816
Extended BKT	0.381	0.355	0.847	0.875	0.406	0.815
$e\Delta_{\varnothing}$	0.368	0.353	0.864	0.891	0.393	0.848
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.353	0.327	0.892	0.913	0.385	0.865
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.402	0.394	0.830	0.845	0.422	0.803

Table 6.3: RMSE and AUC values obtained from learner's performance prediction algorithms on synthetic data. The full dataset consists of 7 transactions per learner, and it is decomposed into training and test sets. Parameter learning is done with cross-validation on the training set, and the performance evaluation is done on the test set.

-0.037, AUC $+0.062$ for $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$).

As reported in Table 6.3, and in accordance with what we observed for the 3-transactions synthetic dataset, the prediction performance of logistic regression models follows the trend described in the literature [Pel17, CPBV19, GKS+20, SWHM22]. IRT is once again the less predicting algorithm on E-PRISM synthetic data, with a large gap (validation RMSE $+0.062$, validation AUC -0.154) with the overall penultimate algorithm which is the $e\Delta$ considering the wrong prerequisite structure. Still, the best logistic regression algorithm Best-LR performs notably great on synthetic data generated from E-PRISM.

Classic and extended BKT algorithms give similar results (validation RMSE 0.408/0.406 for classic/extended BKT). Their performance is lower than $e\Delta_{\varnothing}$ (validation RMSE $+0.013$). Because the synthetic data is generated from the E-PRISM framework, neither slip nor guess phenomena are considered. Thus, it offers an additional constraint to the E-PRISM parameter learning that brings better results for $e\Delta_{\varnothing}$ than for BKT algorithms. In addition, the parameters chosen for generation are similar between $\mathfrak{A}$ and $\mathfrak{B}$. The benefits from the extra features of extended BKT have diminished in these conditions.

Insights from the synthetic data

We conducted experiments to evaluate the performance of several different state-of-the-art algorithms for predicting learner performance on E-PRISM-generated synthetic datasets. The experiments were designed to understand how the different algorithms behave on data tailored for E-PRISM and how they handle an existing correlation between the masteries of two KCs.

We found that the best logistic regression algorithms yield good results for predicting the learner's performance from these synthetic data and may even outperform E-PRISM. The fact that other learners' performance prediction algorithms behave correctly gives a glimpse of the plausibility of the data generated from E-PRISM. Still, when considering enough transactions such that prerequisite relationships exhibit real implications, E-PRISM performs the best compared to the other studied predicting algorithms. Also, classic and extended BKT algorithms show similar trends with E-PRISM when no prerequisite relationships are considered, as they are equivalently structured.

Finally, the results in Table 6.3 confirm that RMSE and AUC values follow the same trend. Then, from this point, we will uniquely consider RMSE values in the next experiments, as they are sufficient to account for the performance of the predicting algorithms.

6.1.3 Predicting the learners' performance from real-world data

We have studied the performance of the different learners' performance prediction algorithms on synthetic data. We now examine the performance of these algorithms on real-world data. Previous experiments were typically focused on using synthetic data to evaluate the performance of these algorithms. Still, it is important also to consider their performance on more realistic, complex datasets. For instance, real-world data imply new phenomena, such as slipping and guessing, described in Bayesian Knowledge Tracing (BKT). Because the E-PRISM learner model doesn't consider any of these, the following analysis results will provide insight into the strengths and limitations of E-PRISM with real-world data compared to other learners' performance prediction algorithms.

Presentation of the studied real-world datasets

In this experiment, we use four real-world datasets to evaluate the performance of the learner's performance prediction algorithms. These datasets were selected because they decompose the domain knowledge into KCs more or less similar to those expected in the E-PRISM framework. Studying these datasets should provide valuable insights into the behavior of the different algorithms regarding the granularity of the KC decomposition. It should allow us to understand the strengths and limitations of these algorithms with real-world data. We represent in Table 6.4 the characteristics of these datasets.

ASSISTments12 and *ASSISTments17* are datasets issued from the ASSISTment system

introduced by Heffernan [FHK09]. ASSISTments is an ITS developed by researchers at Worcester Polytechnic Institute to help students learn math [HH14]. It uses machine learning algorithms and cognitive diagnosis methods to build a learner model and adapt the instruction to the student's needs and learning preferences [FH06]. It is the biggest publicly available knowledge tracing data source [PBH+15]. The granularity of the domain knowledge decomposition differs a little from the granularity suggested in the KLI framework [KCP12], which assumes the KCs to be unitary tasks – see Definition 2.1.2. Still, *ASSISTments12* uses finer KCs than *ASSISTments17*, which relies on wide skills. We remark that some transactions are not labeled, so we can wonder about the correctness of labeled learner interactions.

The *Eedi2020* dataset has been publicly released in a NeurIPS2020 challenge [WLS+21]. It is issued from the Eedi system, discussed in Section 1.1.3. We use the dataset associated with Tasks 1 & 2 of the challenge. The learner traces are collected on multiple-choice questions, and each of these is associated with KCs. The Eedi dataset introduces KCs with four different granularities. The finest granularity corresponds to the KLI framework. In this study, only this granularity is studied.

Finally, the *Kartable* dataset has been provided by Kartable, presented in Section 1.1.4. The data is collected from the exercises available on the Kartable[1] website. They also are multiple-choice questions for the most part. The data has been restricted to the exercises about the chapter "Quadratic Equations and Functions" studied by *Première* (11th grade) french students. This dataset is not freely available.

Dataset	*ASSISTments12*	*ASSISTments17*	*Eedi2020*	*Kartable*
Number of transactions	6 116 189	934 640	165 900	146 339
Number of learners	46 667	1709	2000	7368
Number of KCs	199	411	280	58
Number of items	179 173	3162	22894	312
Mean number of transactions per learner	48	439	56	6
Mean number of items per KC	900	7.69	81.8	5.4
Mean % of correct answers	67.7%	37.3%	68.6%	73,6%

Table 6.4: Characteristics of the real-world datasets

[1] www.kartable.fr

Extracting sub-datasets from the real-world datasets

Because of the E-PRISM computational constraints presented in Chapter 4, we cannot study these real-world datasets as their whole. Consequently, we decompose the datasets into sub-datasets. We describe the procedure for constructing and selecting the sub-datasets in the following.

For each dataset, we consider the set of sub-datasets restricted to pairs of KCs of the original dataset. We focus on the correlation between the masteries of two KCs. Thus, we expect each sub-dataset to only contain the transactions from learners that have trained the two KCs associated with the sub-dataset. Because original real-world datasets rely on large numbers of KCs, the number of sub-datasets for each dataset is substantial. For instance, *ASSISTments12* leads to $39,505$ subdatasets. For each sub-dataset, by denoting $\mathfrak{A}$ and $\mathfrak{B}$ the knowledge components of the sub-dataset, we must study the three configurations of the relationship between $\mathfrak{A}$ and $\mathfrak{B}$ to analyze it, namely $\varnothing$, $\mathfrak{A} \to \mathfrak{B}$, and $\mathfrak{B} \to \mathfrak{A}$. This is why we have selected a restricted number of sub-datasets. We assume the impact of the potential prerequisite relationship between $\mathfrak{A}$ and $\mathfrak{B}$ is more noticeable with many learner transactions with the system. Therefore, for each real-world dataset, we have selected the 6 sub-datasets with the greatest number of learner transactions, considering learners that have practiced the two knowledge components.

The selected sub-datasets are presented in Tables 6.5, 6.6, 6.7, and 6.8. We detail the characteristics of these datasets in Appendix B.

$\mathfrak{A}$	$\mathfrak{B}$
Addition and Subtraction Integers (ASI)	Multiplication and Division Integers (MDI)
Addition and Subtraction Fractions (ASF)	Multiplication Fractions (MF)
Addition and Subtraction Integers (ASI)	Addition and Subtraction Fractions (ASF)
Addition and Subtraction Positive Decimals (ASPD)	Multiplication and Division Positive Decimals (MDPD)
Addition and Subtraction Fractions (ASF)	Division Fractions (DF)
Addition and Subtraction Integers (ASI)	Exponents (E)

Table 6.5: Studied pairs of knowledge components in the *ASSISTments12* dataset

$\mathfrak{A}$	$\mathfrak{B}$
Probability (P)	Equation solving (ES)
Pattern finding (PF)	Probability (P)
Equivalent fractions, decimals, percents (EFDP)	Probability (P)
Pattern finding (PF)	Equation solving (ES)
Area (A)	Probability (P)
Pythagorean theorem (PT)	Probability (P)

Table 6.6: Studied pairs of knowledge components in the *ASSISTments17* dataset

$\mathfrak{A}$	$\mathfrak{B}$
Factors and Highest Common Factor (FHCF)	Multiples and Lowest Common Multiple (MLCM)
Factors and Highest Common Factor (FHCF)	Prime Numbers and Prime Factors (PNPF)
Multiples and Lowest Common Multiple (MLCM)	Prime Numbers and Prime Factors (PNPF)
Volume of Non-Prisms (VNP)	Mental Multiplication and Division (MMD)
Volume of Non-Prisms (VNP)	Mental Addition and Subtraction (MAS)
Mental Addition and Subtraction (MAS)	Mental Multiplication and Division (MMD)

Table 6.7: Studied pairs of knowledge components in the *Eedi2020* dataset

$\mathfrak{A}$	$\mathfrak{B}$
Determine the canonical form of a quadratic polynomial (CF)	Give the roots of a quadratic polynomial (Solve)
Determine if a real number is a root of a quadratic polynomial (Root)	Find an obvious root for a quadratic polynomial (OR)
Give the roots of a quadratic polynomial (Solve)	Determine if a real number is a root of a quadratic polynomial (Root)
Determine the canonical form of a quadratic polynomial (CF)	Give the sign chart of a quadratic polynomial (Chart)
Give the roots of a quadratic polynomial (Solve)	Give the sign chart of a quadratic polynomial (Chart)
Find an obvious root for a quadratic polynomial (OR)	Calculate the discriminant of a quadratic polynomial given in the expanded form (D)

Table 6.8: Studied pairs of knowledge components in the *Kartable* dataset

Finally, we restrict each of these sub-datasets to 7 transactions per learner, as we set the limit to perform E-PRISM parameter learning in a tractable way. The extended variant of BKT will not be studied on real-world sub-datasets, as items must be the same between the train and validation sets. Because of the high variety of items in these datasets, this condition is too complicated to meet.

For each sub-datasets, we perform parameter learning on three E-PRISM learner models ($e\Delta$), each considering a prerequisite relationship configuration between the two KCs. We perform 100 runs of the parameter learning algorithm with a single instance of EM algorithm and $N_{\text{Gibbs}} = 20$, and $M = 0$. We have chosen to run 100 times the parameter learning procedure and not to consider the smoothing allowed by N_{EM} simultaneous EM instances because real-world sub-datasets are larger than synthetic datasets.

As stated in Section 6.1.2, we only study the RMSE values as they are sufficient to understand the performance of algorithms in predicting the learner's answers.

Performance on *ASSISTments12* sub-datasets

Table 6.9b shows the best RMSE values obtained during parameter learning with a 5-fold cross-validation for each sub-dataset of *ASSISTments12*. Table 6.9b relates the RMSE values computed on the validation set of each sub-dataset of *ASSISTments12*. We recall which KCs are defined as KCs $\mathfrak{A}$ and $\mathfrak{B}$ in the header of Tables 6.9a and 6.9b. This defines which prerequisite relationship is considered in $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ and $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$.

First, we analyze the performances of the E-PRISM learner models ($e\Delta$), and we compare them with each other. We can observe in Table 6.9a three possible behavior of the E-PRISM framework. The sub-datasets *(Addition and Subtraction Integers / Exponents)* and *(Addition and Subtraction Positive Decimals / Multiplication and Division Positive Decimals)* show better performance from $e\Delta_{\varnothing}$ than from the other $e\Delta$s. In the sub-datasets *(Addition and Subtraction Fractions / Multiplication Fractions)* and *(Addition and Subtraction Integers / Addition and Subtraction Fractions)*, $e\Delta_{\mathfrak{B}\to\mathfrak{A}}$ and $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ has respectively the best results among the other $e\Delta$s, and the $e\Delta$ considering the other direction for the prerequisite relationship leads to the poorest results. On the contrary, sub-datasets *(Addition and Subtraction Integers / Multiplication and Division Integers)* and *(Division Fractions / Addition and Subtraction Fractions)* show that both $e\Delta$s considering a prerequisite relationship outperform $e\Delta_{\varnothing}$, with one of the possible directions being more performant than the other ($e\Delta_{\mathfrak{B}\to\mathfrak{A}}$ for the two sub-datasets).

We can see in Table 6.9b that the gaps between the E-PRISM learner models ($e\Delta$)s performances observed on the training RMSE values decrease significantly on the validation set. Indeed, only *(Addition and Subtraction Integers / Multiplication and Division Integers)* and *(Division Fractions / Addition and Subtraction Fractions)* allow naming a slightly more performant $e\Delta$. We remark that these two sub-datasets were those where $e\Delta_{\varnothing}$ was the least performant $e\Delta$. Also, because the RMSE values obtained during the training process are lower than

$\mathfrak{A}$	MDI	MF	ASI	ASPD	DF	E
$\mathfrak{B}$	ASI	ASF	ASF	MDPD	ASF	ASI
IRT	0.334	0.429	0.427	0.373	0.440	0.372
PFA	0.339	0.416	0.405	0.364	0.432	0.396
DAS3H	0.327	0.413	0.406	0.354	0.427	0.374
Best-LR	0.333	0.408	0.398	0.367	0.422	0.384
BKT	0.332	0.429	0.411	0.376	0.435	0.389
$e\Delta_{\varnothing}$	0.345	0.425	0.417	0.372	0.446	0.389
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.337	0.426	0.392	0.379	0.439	0.396
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.327	0.420	0.419	0.374	0.434	0.401

(a) Best train RMSE values

$\mathfrak{A}$	MDI	MF	ASI	ASPD	DF	E
$\mathfrak{B}$	ASI	ASF	ASF	MDPD	ASF	ASI
IRT	0.367	0.445	0.404	0.392	0.430	0.393
PFA	0.359	0.434	0.403	0.384	0.418	0.406
DAS3H	0.349	0.429	0.392	0.381	0.416	0.389
Best-LR	0.346	0.424	0.392	0.380	0.413	0.388
BKT	0.360	0.435	0.404	0.384	0.421	0.405
$e\Delta_{\varnothing}$	0.369	0.442	0.411	0.387	0.425	0.409
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.367	0.440	0.411	0.386	0.424	0.410
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.366	0.440	0.411	0.386	0.423	0.409

(b) Validation RMSE values

Table 6.9: Training and validation RMSE values obtained during the process learning process on the sub-datasets derived from *ASSISTments12*. Each sub-dataset consists of at most 7 transactions per learner, and it is decomposed into training and validation sets. Parameter learning is done with 5-fold cross-validation on the training set, and the validation RMSE value is computed on the validation set.

those computed on the validation set, we can assert that E-PRISM slightly overfits.

We now compare the performance of the best $e\Delta$ with the other predicting algorithms. First,

in Table 6.9a, the order of logistic regression models performance reported in Section 6.1.2 slightly varies. For some sub-datasets, DAS3H leads to better training results than Best-LR. This report is not accurate anymore when considering the validation RMSE values in Table 6.9b, where Best-LR is always the best predicting algorithm. In Table 6.9a, we also notice that the RMSE values of E-PRISM obtained during training are approximately included between the RMSE values of PFA and DAS3H/Best-LR. Moreover, for the sub-dataset *(Addition and Subtraction Integers / Addition and Subtraction Fractions)*, we even have $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ being the most performant prediction algorithm. However, these results don't replicate when considering the RMSE values on the validation set. We can observe in Table 6.9b that the validation RMSE values of E-PRISM are higher than those of the best logistic regression models. Thus, E-PRISM shows a higher tendency to overfit than the other algorithms.

Performance on *ASSISTments17* sub-datasets

Table 6.10b presents the training RMSE values computed from the dataset *ASSISTments17*, using 5-fold cross-validation. Table 6.10b shows the RMSE values computed on the validation sets.

First, we can notice that the computed values of RMSE are way higher than those computed for *ASSISTments12*. *ASSISTments17* dataset leads to the poorest prediction algorithm performance. We compare E-PRISM learner models ($e\Delta$) with each other and then with the other predicting algorithms.

We observe in 6.10a that for each sub-dataset, $e\Delta_{\varnothing}$ leads to a lower training RMSE value than the other $e\Delta$s, except for the sub-dataset *(Pythagorean theorem / Probability)* where $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$ outperforms the others. On the contrary, Table 6.10b shows that the validation RMSE gives different results. Still, we observe the same gap reduction phenomenon we saw on *ASSISTments12* data, and the differences between $e\Delta$ validation RMSE values are too small to be significant for *ASSISTments17*.

Similarly to the results on *ASSISTments12* sub-datasets, DAS3H and Best-LR are the best logistic regression models. Which algorithm is the best between the two depends on the pair of KCs according to training RMSE values, while it is always Best-LR according to validation RMSE values. This suggests that DAS3H overfits more than Best-LR on the data of both *ASSISTments12* and *ASSISTments17*. Nevertheless, both outperform other predicting algorithms, including every $e\Delta$.

Finally, we observe in both Tables 6.10a and 6.10b that IRT outperforms PFA. Generally, IRT shows better results compared to the other algorithms than it has shown until now. On the contrary, PFA and BKT offer poor results regarding those observed until now. The poor results of both PFA and BKT can be explained by the granularity of KCs described in the *ASSISTments17* dataset. IRT better handles larger concepts as knowledge components than PFA, BKT, and E-PRISM.

$\mathfrak{A}$	P	P	P	PF	P	PT
$\mathfrak{B}$	ES	PF	EFDP	ES	A	P
IRT	0.460	0.470	0.475	0.463	0.452	0.475
PFA	0.487	0.478	0.487	0.471	0.470	0.483
DAS3H	0.450	0.468	0.463	0.456	0.453	0.466
Best-LR	0.458	0.465	0.477	0.452	0.462	0.472
BKT	0.481	0.480	0.491	0.469	0.474	0.485
$e\Delta_{\varnothing}$	0.482	0.478	0.485	0.469	0.468	0.490
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.484	0.483	0.487	0.477	0.468	0.486
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.483	0.484	0.488	0.475	0.468	0.489

(a) Training RMSE values

$\mathfrak{A}$	P	P	P	PF	P	PT
$\mathfrak{B}$	ES	PF	EFDP	ES	A	P
IRT	0.470	0.482	0.472	0.475	0.470	0.484
PFA	0.488	0.484	0.486	0.487	0.479	0.488
DAS3H	0.465	0.479	0.474	0.473	0.464	0.483
Best-LR	0.461	0.465	0.470	0.473	0.465	0.483
BKT	0.488	0.482	0.482	0.488	0.479	0.487
$e\Delta_{\varnothing}$	0.496	0.483	0.488	0.488	0.483	0.487
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.496	0.486	0.489	0.492	0.483	0.486
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.495	0.482	0.487	0.490	0.482	0.492

(b) Validation RMSE values

Table 6.10: Validation RMSE values obtained during the process learning process on the sub-datasets derived from *ASSISTments17*. Each sub-dataset consists of at most 7 transactions per learner, and it is decomposed into training and validation sets. Parameter learning is done with 5-fold cross-validation on the training set, and the validation RMSE value is computed on the validation set.

Performance on *Eedi2020* sub-datasets

Table 6.11b shows the best training RMSE values obtained from a 5-fold cross-validation learning process for each selected sub-dataset of *Eedi*. Table 6.11b reports the validation

RMSE values.

$\mathfrak{A}$	MLCM	PNPF	PNPF	MMD	MAS	MMD
$\mathfrak{B}$	FHCF	FHCF	MLCM	VNP	VNP	MAS
IRT	0.448	0.437	0.465	0.408	0.417	0.418
PFA	0.451	0.445	0.464	0.399	0.413	0.407
DAS3H	0.440	0.434	0.455	0.377	0.412	0.411
Best-LR	0.430	0.442	0.464	0.399	0.398	0.418
BKT	0.444	0.442	0.466	0.413	0.410	0.415
$e\Delta_{\varnothing}$	0.449	0.455	0.460	0.409	0.396	0.414
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.449	0.444	0.461	0.402	0.409	0.402
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.450	0.450	0.470	0.399	0.403	0.408

(a) Training RMSE values

$\mathfrak{A}$	MLCM	PNPF	PNPF	MMD	MAS	MMD
$\mathfrak{B}$	FHCF	FHCF	MLCM	VNP	VNP	MAS
IRT	0.456	0.448	0.467	0.444	0.408	0.417
PFA	0.475	0.450	0.470	0.444	0.409	0.416
DAS3H	0.467	0.443	0.464	0.441	0.409	0.401
Best-LR	0.460	0.436	0.456	0.439	0.408	0.408
BKT	0.479	0.449	0.470	0.448	0.411	0.418
$e\Delta_{\varnothing}$	0.482	0.449	0.476	0.446	0.414	0.422
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.478	0.448	0.470	0.443	0.418	0.423
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.477	0.447	0.472	0.445	0.414	0.422

(b) Validation RMSE values

Table 6.11: Validation RMSE values obtained during the process learning process on the sub-datasets derived from *Eedi2020*. Each sub-dataset consists of at most 7 transactions per learner, and it is decomposed into training and validation sets. Parameter learning is done with 5-fold cross-validation on the training set, and the validation RMSE value is computed on the validation set.

As we deplored for *ASSISTments17*, the RMSE values computed on the dataset *Eedi* are relatively high, as reported in Table 6.11. We can see in the training RMSE values in Table 6.11a that E-PRISM overall outperforms both IRT and BKT. $e\Delta_{\varnothing}$ outclasses the other

$e\Delta$s on datasets *(Factors and Highest Common Factor / Multiples and Lowest Common Multiple)*, *(Multiples and Lowest Common Multiple / Prime Numbers and Prime Factors)*, and *(Volume of Non-Prisms / Mental Addition and Subtraction)*. The other sub-datasets show better results for a $e\Delta$ considering the existence of a prerequisite relationship. Nevertheless, the differences between RMSE values of the $e\Delta$s are small, and they are not observed anymore on validation RMSE values. Table 6.11b shows that the RMSE values are significantly higher than those obtained during training. This report suggests that E-PRISM undergoes a consequential overfit. DAS3H and Best-LR outperform again on training and validation data. Depending on the sub-dataset, they alternate being the most performant predicting algorithm.

Performance on *Kartable* sub-datasets

Finally, we represent in Table 6.12b the best training RMSE values obtained from a 5-fold cross-validation learning process for each selected sub-dataset of *Kartable*. The RMSE values obtained on the validation set are reported in Table 6.12b.

According to Table 6.12a, $e\Delta_{\varnothing}$ is the most performant E-PRISM model on training for every sub-dataset but *(Determine if a real number is a root of a quadratic polynomial / Find an obvious root for a quadratic polynomial)* and *(Give the roots of a quadratic polynomial / Give the sign chart of a quadratic polynomial)*. Nevertheless, these results do not hold on to the validation set for which the RMSE values are, for the most part, similar between all $e\Delta$s, as reported in Table 6.12b.

RMSE values from the training and validation sets show similar results for the other predicting algorithms. Best-LR and DAS3H are the best predicting algorithms, with the former slightly outperforming. The other logistic regression models and BKT show RMSE values close to the E-PRISM learner models.

6.1.4 Discussion

We can elicit two conclusions from the results reported in Sections 6.1.2 and 6.1.3 about the performance of E-PRISM. We compared the E-PRISM learner model with other learners' performance prediction algorithms on synthetic and real-world data. We also studied the difference in the performance of the different E-PRISM learner models ($e\Delta$) to highlight the importance of considering the correct prerequisite relationship.

Overall performance of E-PRISM models

We have reported that E-PRISM gives similar results to PFA on training and IRT on validation sets. It follows the same trend as the other learner's performance prediction algorithms. In particular, we have observed the RMSE values get close to 0.5, which is the maximal value of RMSE, on some datasets, such as *Eedi* and *ASSISTments17*. Nevertheless, it performs well on the other datasets *ASSISTments12* and *Kartable*. We believe these

$\mathfrak{A}$	CF	Root	Solve	CF	Solve	OR
$\mathfrak{B}$	Solve	OR	Root	Chart	Chart	D
IRT	0.418	0.360	0.356	0.400	0.373	0.294
PFA	0.418	0.347	0.359	0.404	0.386	0.285
DAS3H	0.410	0.339	0.354	0.387	0.367	0.299
Best-LR	0.415	0.355	0.354	0.396	0.359	0.290
BKT	0.414	0.363	0.363	0.405	0.379	0.301
$e\Delta_{\varnothing}$	0.421	0.372	0.359	0.389	0.389	0.286
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.424	0.359	0.366	0.407	0.376	0.296
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.422	0.364	0.364	0.398	0.368	0.294

(a) Training RMSE values

$\mathfrak{A}$	CF	Root	Solve	CF	Solve	OR
$\mathfrak{B}$	Solve	OR	Root	Chart	Chart	D
IRT	0.437	0.368	0.361	0.410	0.388	0.317
PFA	0.438	0.366	0.370	0.411	0.394	0.312
DAS3H	0.429	0.361	0.358	0.405	0.385	0.309
Best-LR	0.426	0.361	0.359	0.404	0.384	0.307
BKT	0.439	0.366	0.369	0.411	0.393	0.312
$e\Delta_{\varnothing}$	0.441	0.368	0.370	0.412	0.395	0.318
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.439	0.371	0.370	0.412	0.394	0.316
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.440	0.368	0.370	0.413	0.395	0.318

(b) Validation RMSE values

Table 6.12: Validation RMSE values obtained during the process learning process on the sub-datasets derived from Kartable. Each sub-dataset consists of at most 7 transactions per learner, and it is decomposed into training and validation sets. Parameter learning is done with 5-fold cross-validation on the training set, and the validation RMSE value is computed on the validation set.

differences are due to the granularity of the KC decomposition. The great performance of IRT on datasets *ASSISTments17* and *Eedi* compared to the other logistic regression models acknowledges this hypothesis.

Difference between results on synthetic and real-world datasets

We also observed a gap between the performances of E-PRISM on synthetic and real-world data. This sounds logical as the synthetic data is tailored to fit E-PRISM models. The fact that the E-PRISM models even outperform BKT models on synthetic data enlighted that no slip nor guess phenomenon is represented in the synthetic data. This is the main pain point of our model, as real-world data relate to noisy observables. These reasons explain the big difference between the performance of E-PRISM on synthetic and real-world data.

In every real-world sub-dataset, we have observed a significant difference between the performance of E-PRISM models on the test and the validation sets. This gap reveals that E-PRISM tends to overfit from the training data. This report feels legit, as the non-consideration of slip and guess phenomena inevitably implies a deeper bond between the observed values and the mastery variables by assuming no noise in the data.

Comparison between instances of E-PRISM

Last but not least, the different instances of E-PRISM led to various results that may inform the prerequisite structure of the latent domain model. For most sub-datasets, the best E-PRISM learner model did not consider any prerequisite relationships between the two KCs. Nevertheless, for some pairs of KCs, E-PRISM learner models supposing the existence of a prerequisite relationship outperformed.

For instance, the E-PRISM learner model considering the prerequisite relationship *Addition and Subtraction Integers* $\rightarrow$ *Multiplication and Division Integers* showed better results than the other $e\Delta$s on both test and validation sets. On the other hand, an instance often showed tremendous results compared to the others but uniquely on the training RMSE values. For instance, in the *Kartable* dataset, we observe that the most performing $e\Delta$s consider relationships like *Give the sign chart of a quadratic polynomial* $\rightarrow$ *Give the roots of a quadratic polynomial* or *Determine if a real number is a root of a quadratic polynomial* $\rightarrow$ *Find an obvious root for a quadratic polynomial*. These prerequisite relationships look plausible and should be further investigated.

$\mathcal{M}_1$ metric related to the RMSE value

It may be challenging to determine the existence of a prerequisite relationship between two KCs by only studying the values of RMSE obtained during both training and validation. Still, studying the prerequisite relationships from a given dataset should not care about overfitting the parameter learning, as the wanted result is the prerequisite structure of KCs expressed in the data, more than its generalization to new data. Then, the unique value to keep in mind when studying the prerequisite relationships with E-PRISM is the training RMSE (or the NLL, which is directly related).

On the training data, we use the RMSE metric to identify the existence and direction of the

prerequisite relationship. Indeed, by comparing the RMSE values obtained from the different E-PRISM learner models, we identify which instance of E-PRISM leads to the lowest RMSE value. We can then determine the existence and the direction of the prerequisite relationship. Specifically, we compute $\mathcal{M}_1$, the relative difference between the lowest RMSE value of the considered instance and the lowest RMSE value of the instance without the prerequisite relationship. $\mathcal{M}_1$ can be computed according to Equation 6.1 for each dataset and each direction of the prerequisite relationship. It allows us to quickly and easily determine the best-fit direction of the prerequisite relationship.

$$\mathcal{M}_1(\mathfrak{A} \to \mathfrak{B}) = \frac{1}{K} \frac{(\text{RMSE of } e\Delta_{\varnothing} - \text{RMSE of } e\Delta_{\mathfrak{A}\to\mathfrak{B}})}{\text{RMSE of } e\Delta_{\varnothing}} \tag{6.1}$$

where $\mathfrak{A}$ and $\mathfrak{B}$ are two KCs, and K is a normalizing constant.

$\mathcal{M}_1(\mathfrak{A} \to \mathfrak{B})$ is a measure for the existence of the prerequisite relationship. It indicates how better the E-PRISM model performs by considering $\mathfrak{A} \to \mathfrak{B}$. $\mathcal{M}_1$ ranges from -1 (very unlikely there exists a relationship $\mathfrak{A} \to \mathfrak{B}$) to 1 (very likely there exists a relationship $\mathfrak{A} \to \mathfrak{B}$). We can evaluate the direction of the prerequisite relationship indicated by this metric by comparing $\mathcal{M}_1(\mathfrak{A} \to \mathfrak{B})$ and $\mathcal{M}_1(\mathfrak{B} \to \mathfrak{A})$.

6.2 Study of E-PRISM learner model parameters trained on synthetic data

As mentioned several times, the main benefit of E-PRISM is to integrate the KC prerequisite relationships in modeling the learner's knowledge state. Model parameters in E-PRISM are designed to emphasize the causal effect of the prerequisite relationships that the domain model considers. In this section, we study the value of these parameters obtained from the parameter learning of the E-PRISM learner model. More specifically, we analyze the distribution of the learned parameters.

We first study the value of the learned model parameters from synthetic data to apprehend the behavior of these parameters better. The use of synthetic data allows us to control the prerequisite structure of the domain model and then provide important insights into the values of learned parameters.

6.2.1 Protocol

We define two types of E-PRISM learner models ($e\Delta$) in the following. First, the $e\Delta$ that have been used for generating the synthetic datasets. They will not be used further. Then, the $e\Delta$ used for parameter learning are introduced.

Generative sources of synthetic datasets

We suppose a domain model composed of two KCs $\mathfrak{A}$ and $\mathfrak{B}$. We introduce three synthetic datasets $\mathcal{D}_{\text{no prereq.}}$, $\mathcal{D}_{\text{strong}}$, and $\mathcal{D}_{\text{weak}}$, respectively generated from the E-PRISM models $e\Delta_{\varnothing}$, $e\Delta_{\text{strong}}$, $e\Delta_{\text{weak}}$.

The domain model of $e\Delta_{\varnothing}$ considers no prerequisite relationships. The parameters that rule $e\Delta_{\varnothing}$ are:

- prior parameters: $p_{\mathfrak{A}} = 0.1$ and $p_{\mathfrak{B}} = 0.08$
- learn parameters: $l_{\mathfrak{A}} = 0.3$ and $l_{\mathfrak{B}} = 0.2$
- forget parameters: $f_{\mathfrak{A}} = 0.01$ and $f_{\mathfrak{B}} = 0.04$

The domain models of $e\Delta_{\text{strong}}$ and $e\Delta_{\text{weak}}$ consider the prerequisite relationship $\mathfrak{A} \rightarrow \mathfrak{B}$. The KC parameters (prior, learn, and forget) are the same as the $e\Delta_{\varnothing}$ ones. The prerequisite parameters of $e\Delta_{\text{strong}}$ values:

- q parameter: $q_{\mathfrak{B},\mathfrak{A}} = 0.1$
- s parameter: $s_{\mathfrak{B},\mathfrak{A}} = 0.05$.

The prerequisite parameters of $e\Delta_{\text{weak}}$ are:

- q parameter: $q_{\mathfrak{B},\mathfrak{A}} = 0.4$

- s parameter: $s_{\mathfrak{B},\mathfrak{A}} = 0.35$.

From each $e\Delta$, we model 200 learners and generate the complete dataset corresponding to 7 transactions per learner. Then, we hide the values that should be hidden to correspond to an actual E-PRISM training dataset. Each synthetic dataset is composed of 1400 entries.

E-PRISM instances for parameter learning

Onto each synthetic dataset, we consider two instances of the E-PRISM learner model. Each $e\Delta$ considers either the correct or inverse direction of the prerequisite relationship between $\mathfrak{A}$ and $\mathfrak{B}$. The $e\Delta$ considering the inverse direction will be denoted $e\Delta_{\text{inv.}}$. For each, we proceed 100 runs of the MCEM algorithm with BGS approximate inference with $N_{\text{EM}} = 10$, $N_{\text{Gibbs}} = 20$, and $M = 0$. For each run, we consider the full dataset as the training dataset. The purpose here is not to evaluate the model but to compute the most fitted parameters to the data.

6.2.2 KC parameters learned from synthetic data

Because of the high variability of parameter learning results with our settings, we choose to represent the weighted density plot of each parameter. We weigh the value obtained for each parameter with the normalized NLL obtained by the learned model on the whole data. We represent in Figure 6.1 the weighted density of the learn and forget parameters computed from the 100 runs of parameter learning.

Values of $e\Delta$ considering the correct direction of the prerequisite relationship

Figure 6.1 shows that the distribution of the l and f parameters obtained from training $e\Delta_{\text{strong}}$ correspond to the learn and forget parameter values used for generation. Nevertheless, we notice a slight shift of the $l_{\mathfrak{B}}$ value, which should equal 0.2. The value of $f_{\mathfrak{B}}$ is slightly lower than expected. Overall, the values obtained from training from a dataset exhibiting a strong correlation between two KCs are close to the generation values. The l values learned from $e\Delta_{\text{weak}}$ show a more significant shift compared to generation values: $l_{\mathfrak{A}}$ is higher than the expected values, while the learned value of $l_{\mathfrak{B}}$ is lower than expected. The $f_{\mathfrak{B}}$ value shows a flattened density from 0 to 0.4. When there is a weak prerequisite relationship, the learned value of the $f_{\mathfrak{B}}$ parameter associated with the target KC is less defined than for a strong prerequisite relationship. We assume this means the causal effect of the source KC mastery on the target KC $\mathfrak{B}$ mastery, implied by the learned model, is too significant to represent the actual causality contained in $\mathcal{D}_{\text{weak}}$.

We also observe the distributions of the $l_{\mathfrak{B}}$ value on $e\Delta_{\text{inv. strong}}$ and $e\Delta_{\text{inv. weak}}$ ($e\Delta_{\text{inv.}}$ respectively applied to $\mathcal{D}_{\text{strong}}$ and $\mathcal{D}_{\text{weak}}$) are indeed extremely flattened with values from 0.3 to 0.9. These values are very high for representing the forget phenomenon. On the contrary, the $l_{\mathfrak{A}}$ value is distributed as a narrow peak below the expected value. This result

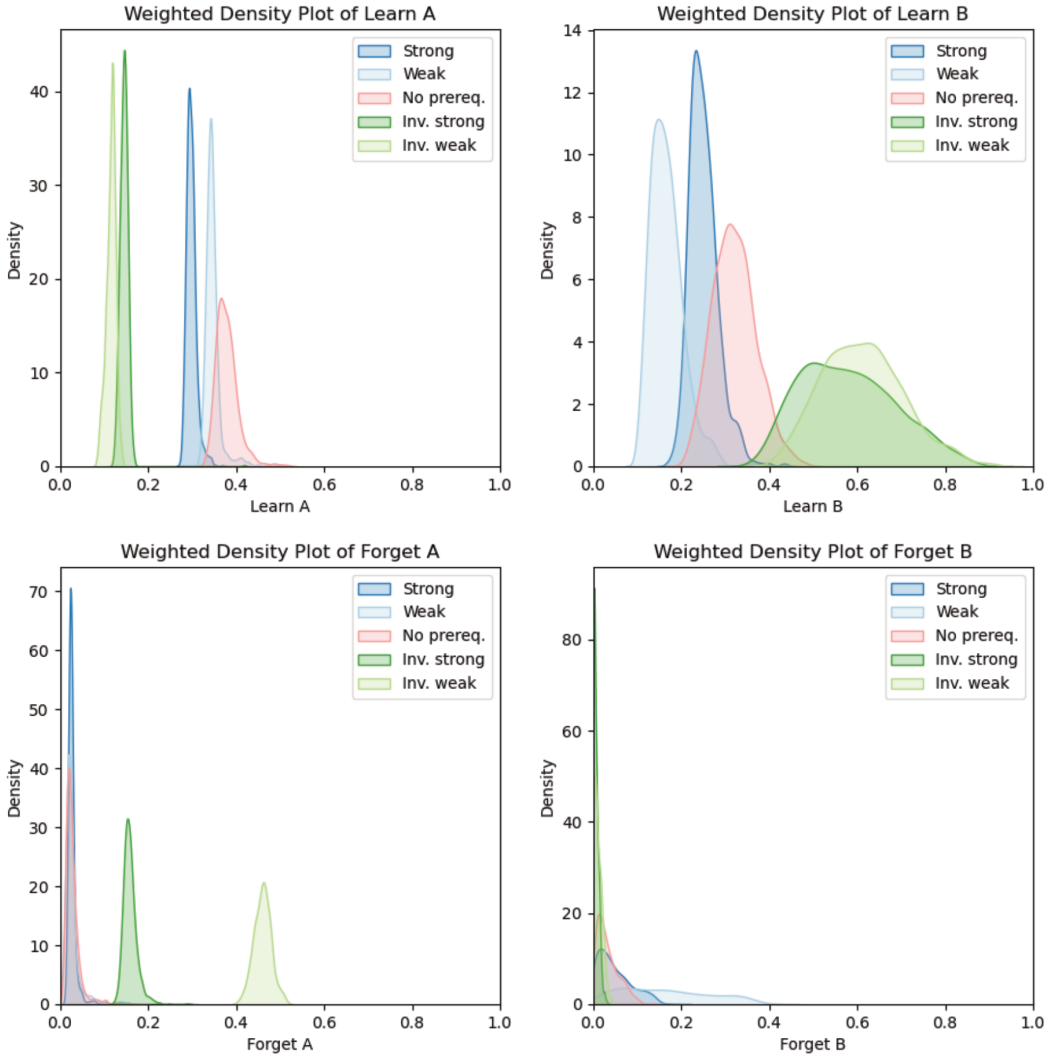


Figure 6.1: Distribution of the "learn" and "forget" parameters' values for $e\Delta$s trained on $\mathcal{D}_{\text{no prereq.}}$, $\mathcal{D}_{\text{weak}}$, and $\mathcal{D}_{\text{strong}}$. The values are learned with MCEM parameter learning with BGS approximate inference.

feels logical, as the causal effect of the prerequisite relationship $\mathfrak{B} \rightarrow \mathfrak{A}$ (which is the causal effect of the mastery of $\mathfrak{B}$ on the mastery of $\mathfrak{A}$) is shifted to the causal effect of the learning of $\mathfrak{A}$.

Values of $e\Delta$ considering the inverse direction of the prerequisite relationship

We also notice that the distribution peak of $f_{\mathfrak{B}}$ is closer to zero when learning $e\Delta_{\text{inv. strong}}$ than when learning on $e\Delta_{\text{strong}}$, and the values of $f_{\mathfrak{A}}$ form peaks with high median values (approximately 0.2 for values learned from $\mathcal{D}_{\text{strong}}$, 0.45 from $\mathcal{D}_{\text{weak}}$). This compensates for the unduly causal effect of the source KC mastery on the target KC mastery considered in the trained E-PRISM model (source and target KCs in the sense of the prerequisite

relationships).

Overall, we remark that the values of the source KC learn parameter and the target KC forget parameter of E-PRISM models that considers the correct direction of the prerequisite relationship are always greater than those of E-PRISM models considering the wrong direction. On the contrary, the values of the target KC learn and the source KC forget parameters of the E-PRISM models that consider the correct direction are consistently lower than those of models about the wrong direction.

Values from $\mathcal{D}_{\text{no prereq.}}$

Finally, the values obtained from learning on the dataset $\mathcal{D}_{\text{no prereq.}}$ also show a slight increase compared to the expected values. However, the learned values of the f parameter equal the values used for generation. The behavior of the distribution of the learned parameter values from $\mathcal{D}_{\text{no prereq.}}$ is quite similar to the distribution of those learned from $\mathcal{D}_{\text{strong}}$, except for the value of $l_{\mathfrak{B}}$. Because a prerequisite relationship is considered during learning, and because it does not exist in the training dataset, the causal effect of $\mathfrak{A}$ on $\mathfrak{B}$ needs to be compensated to correctly represent the mastery of $\mathfrak{B}$.

6.2.3 Prerequisite parameters learned from synthetic data

In the second place, we study the learned values of the prerequisite parameters that should numerically highlight the domain model's prerequisite structure. Figure 6.2 reports the weighted density of the q and s parameters computed from the 100 runs of parameter learning. q and s are supposed to measure the causal effect of $\mathfrak{A}$ on $\mathfrak{B}$. The density is also weighted by the normalized NLL obtained from learning.

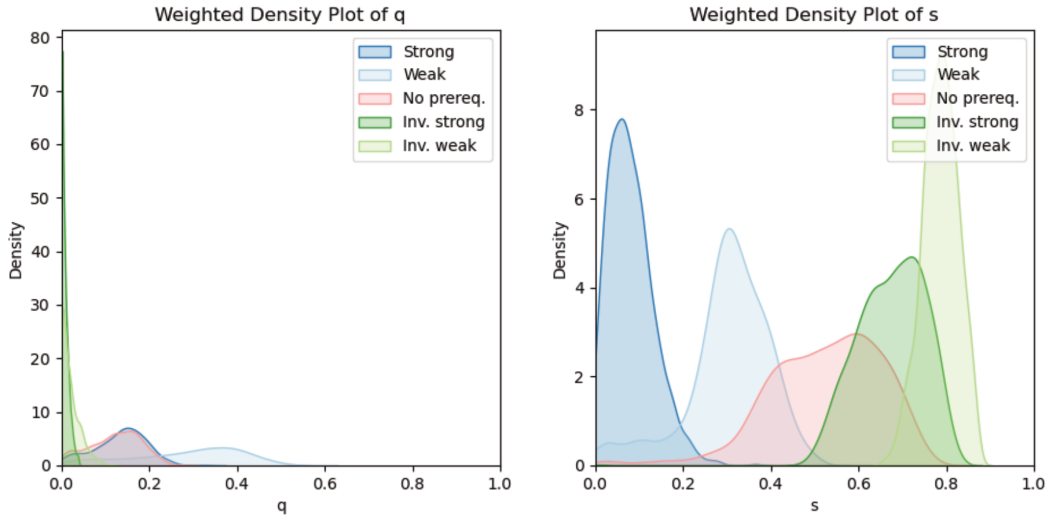


Figure 6.2: Distribution of the q and s parameters' values for $e\Delta$s trained on $\mathcal{D}_{\text{no prereq.}}$, $\mathcal{D}_{\text{weak}}$, and $\mathcal{D}_{\text{strong}}$. The values are learned with MCEM parameter learning with BGS approximate inference.

We observe in Figure 6.2 a non-symmetric behavior of the q and s parameters. The density plots of q from the learning on the different datasets are condensed to low values, while the density plots of the s parameter cover the whole specter of values.

q parameter values

In detail, the q value obtained from $e\Delta_{\text{strong}}$ is close to the generation value. We observe the same trend for the value obtained from $e\Delta_{\text{weak}}$, even if the density plot is more flattened. Learning on $\mathcal{D}_{\text{no prereq.}}$ leads to a q value similar to the one computed with $e\Delta_{\text{strong}}$. On the contrary, the learned values of q are extremely close to zero when the dataset relies on a prerequisite relationship in the inverse direction than the one considered in the training model. We return to the interpretation we provided for the q parameter in Section 4.2.4 to apprehend the signification of these values better. Given a prerequisite relationship, the q parameter associated with the relationship is supposed to represent if the mastery of the source knowledge component is insufficient for the mastery of the target knowledge component. Training a learner model with a domain model considering the wrong direction of the prerequisite relationship in the data would result in a very low value of the q parameter. Consequently, the q values obtained from training $e\Delta_{\text{inv. strong}}$ and $e\Delta_{\text{inv. weak}}$ should be close to zero.

s parameter values

We now focus on the values of the s parameter computed from training on the different datasets. Again, the values obtained from $e\Delta_{\text{strong}}$ and $e\Delta_{\text{weak}}$ are close to the values used for generation. The values of the s parameter computed from $D_{\text{no prereq.}}$ are distributed around 0.5, and it has the most spread distribution. Ultimately, the datasets considering the prerequisite relationship in the inverse direction have led to learned s values greater than 0.5. More surprisingly, the value obtained from $e\Delta_{\text{inv. strong}}$ is lower than the one from $e\Delta_{\text{inv. weak}}$. These results on the s parameter are significant, as they depict s as a clear metric of the prerequisite relationship. We recall that the s parameter associated with a prerequisite relationship is supposed to express the necessity of the mastery of the source KC for the mastery of the target KC. The values of s correspond to the generation values when the prerequisite relationship expressed in the training dataset is the one considered by the training model. When no correlation is expressed in the data, the value of s is close to 0.5, meaning that the mastery of the source KC can either be necessary or not for the mastery of the target KC. Finally, when the prerequisite relationship in the training model is in the wrong direction with regard to the causality considered in the data, the value of s increases to be greater than 0.5, meaning the mastery of the source KC is not necessary to master the target KC.

6.2.4 A promising way to define a prerequisite relationship

In this section, we have studied the distribution of the learned parameters of E-PRISM learner models from several datasets that express different variations of a prerequisite relationship between two knowledge components. Our results open the path for a promising solution for measuring the existence and the strength of a prerequisite relationship between two knowledge components from datasets about learners' transactions.

Upon analyzing these distributions, we have made the following observations:

1. The peak value of q and s in the distribution obtained from parameter learning in the E-PRISM instances considering the correct direction of the prerequisite relationship are approximatively accurate. This suggests that the E-PRISM framework accurately learns the prerequisite parameters when the prerequisite relationship is in the same direction as the one expressed in the synthetic dataset.

2. For the E-PRISM instances considering the wrong direction, the peak value of the q parameter distribution is very close to zero, and the peak value of the s parameter distribution is greater than 0.7.

3. The values of s in the distribution obtained from the instance considering the correct direction has non-zero density for $s = 0$. It means that there is at least one training execution that learns the value 0 for s.

4. We observe clear differences in prerequisite parameter values whether the direction considered by the instance of E-PRISM is different from the one expressed in the synthetic dataset:

    - The peak value of the q parameter in the distribution of the instance considering the correct direction is greater than that obtained from the instance considering the wrong direction.

    - The peak value of the s parameter in the distribution of the instance considering the correct direction is much lower than that obtained from the instance considering the wrong direction.

6.3 Study of learned parameter distribution from real-world data

In this section, we study the parameter distribution obtained from each real-world datasets presented in Section 6.1.3. In the following, we plot the distribution of the parameters' value obtained from learning on the different datasets. For representing the values obtained from the learning of an E-PRISM instance $e\Delta_{\mathfrak{A}\to\mathfrak{B}}$, we call $\mathfrak{A}$ the source KC and $\mathfrak{B}$ the target KC.

6.3.1 KC parameters learned from real-world data

In the following, we examine the KC parameters' density plots. KC parameters are the l and f parameters. Then for each sub-dataset, we represent the density plots of the learned l and f parameters from the two E-PRISM models, with different directions of the prerequisite relationship between the two KCs. Through this comparison, we aim to identify any distinctions or similarities between the learned parameter distributions and gain insight into the underlying structure of the model and its ability to capture the relationships between different knowledge components in different directions.

KC parameters computed from *ASSISTments12* sub-datasets

We represent in Figure 6.3 the density plot of the KC parameters of E-PRISM learner models ($e\Delta$) induced by each possible configuration of prerequisite relationships.

The results indicate that the peak values of the l_{source} parameter range approximately from 0.42 to 0.7 for all E-PRISM models, with an average of around 0.58. Similarly, the peak values of the l_{target} parameter range from 0.52 to 0.76, with an average of around 0.64. Then, we notice that all the values of the l parameter obtained from training represented in Figure 6.3 are greater than 0.4. We observe a similar phenomenon with all the values of the f parameter, which are approximately lower than 0.2. The peak values of the f_{source} parameter indeed range from 0.09 to 0.21, with an average of around 0.14, while the values of the f_{target} parameter range from 0.04 to 0.1, with an average of around 0.07. We also observe that the average of the peak values of l_{target} is higher than l_{source}, while the peak values of the f parameter are lower for the target KC than for the source KC.

KC parameters computed from *ASSISTments17* sub-datasets

In Figure 6.3, we represent the density plot of the different directions of prerequisite relationships in the *ASSISTments17* sub-datasets.

We first notice in Figure 6.3 that the l_{source} parameter values obtained from training are evenly distributed between 0.2 and 0.6, and l_{target} parameter values are for the most contained between 0.6 and 0.8, with exceptions for (*Probability* $\to$ *Area*), (*Probability* $\to$ *Equiv*), and (*Probability* $\to$ *Pattern finding*) which show lower values.

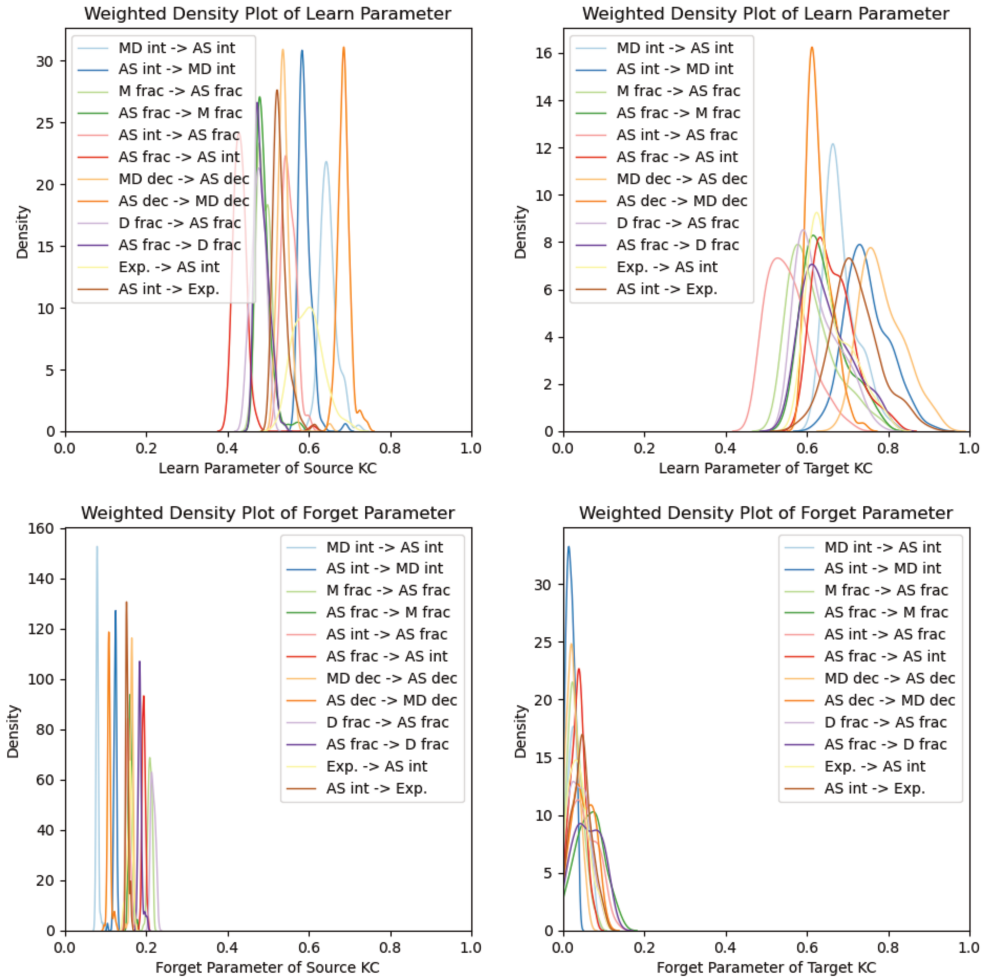


Figure 6.3: Density plots of the learned values of the l and f parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *ASSISTments12* sub-datasets.

We also observe that the values of f_{source} parameters are included from 0.2 to 0.4. We still remark that (*Area* $\rightarrow$ *Probability*) and (*Pythagore theorem* $\rightarrow$ *Probability*) have greater values. The f_{target} values are majorly lower than 0.2. Once again, sub-datasets *(Probability / Area)* and *(Pythagore theorem / Probability)* show different values, greater than the others.

We point out the central role of the KC *Probability* in borderline cases. As we do not have access to the resources associated with this KC on which *ASSISTments17* data is collected, we cannot analyze the granularity of KC defined in the datasets. Still, the name attached with the KC *Probability* seems to encompass many KCs if they were defined as in Definition 2.1.2.

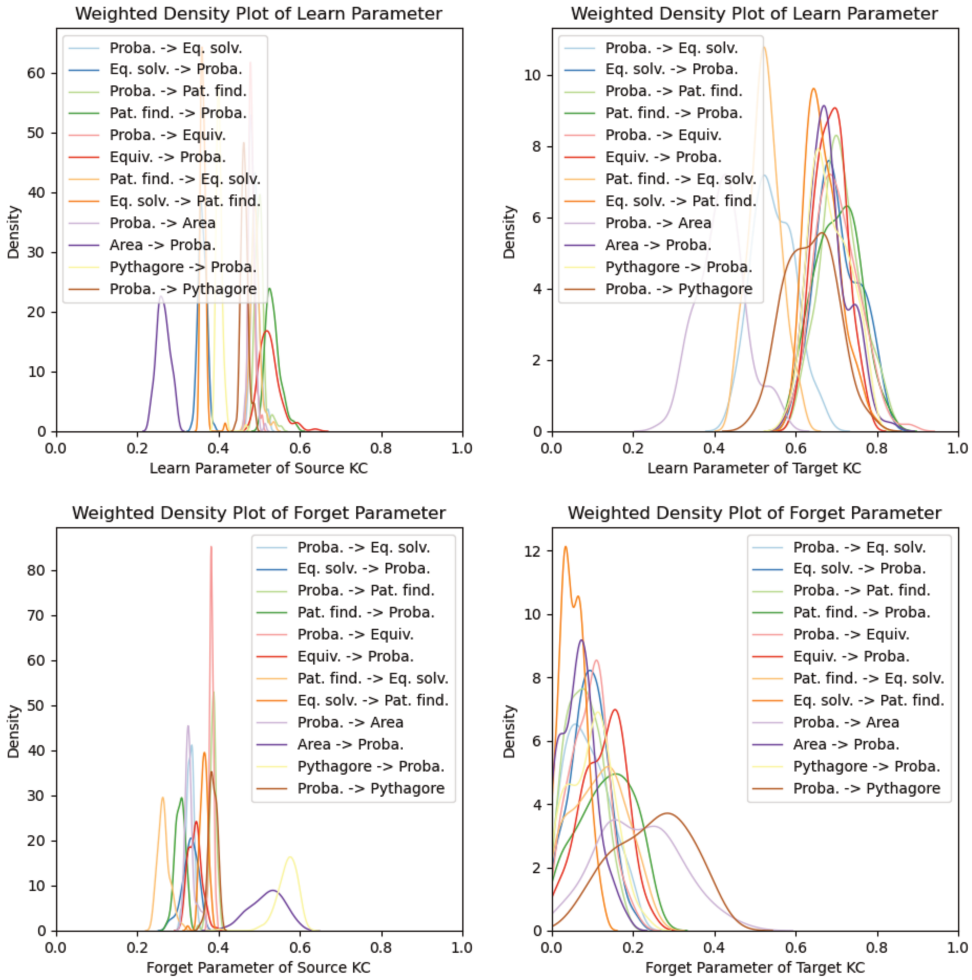


Figure 6.4: Density plots of the learned values of the l and f parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *ASSISTments17* sub-datasets.

KC parameters computed from *Eedi2020* sub-datasets

We represent in Table 6.5 the l and f density plots for sub-datasets of the *Eedi2020* dataset.

For each subset of the data, we compare the peak values of the parameter distributions spotted in Figure 6.5. Our analysis shows that the model's performance varies across the different subsets of data. For example, when the sub-dataset *(Factors and Highest Common Factor / Multiples and Lowest Common Multiple)* is considered, the peak values of the l_{source} and f_{source} parameters are higher for *Multiples and Lowest Common Multiple* $\rightarrow$ *Factors and Highest Common Factor*. The E-PRISM model that considers *Prime Numbers and Prime Factors* $\rightarrow$ *Factors and Highest Common Factor* shows higher peak values for the l_{source} and f_{source} parameters than the one considering *Factors and Highest Common Factor* $\rightarrow$ *Prime Numbers*

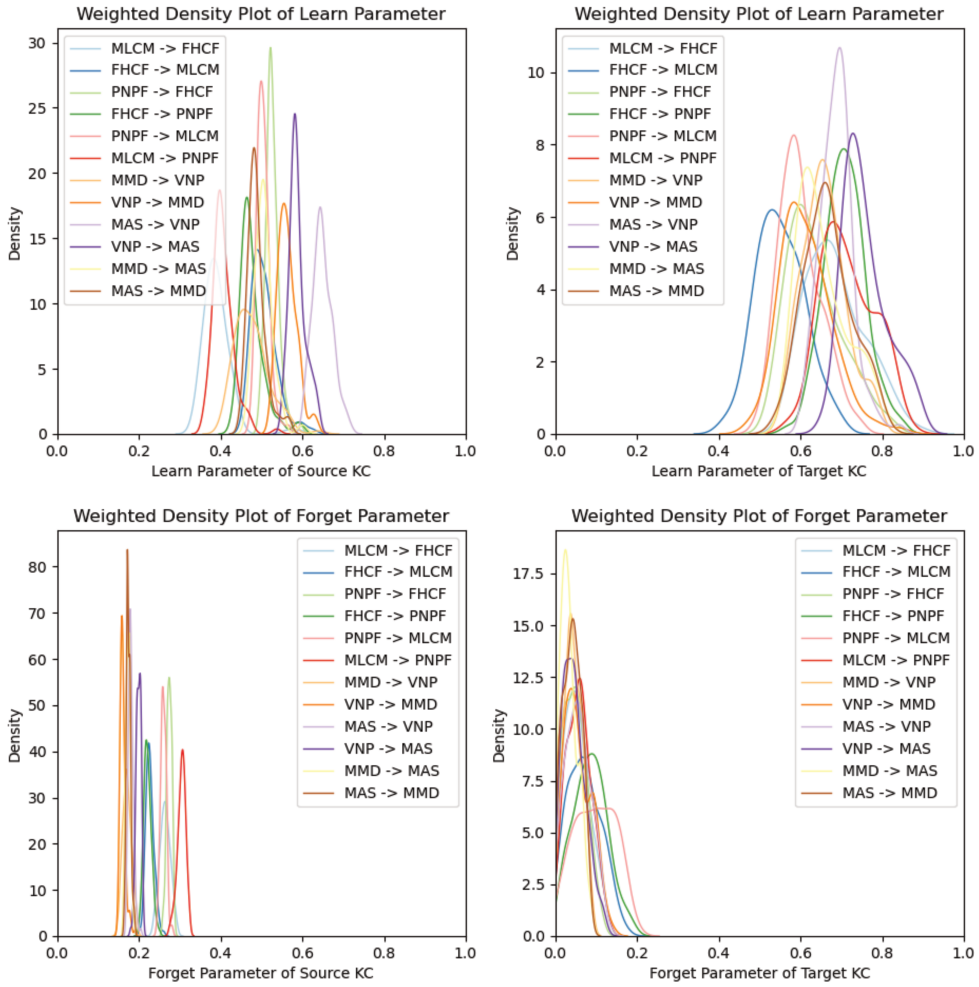


Figure 6.5: Density plots of the learned values of the l and f parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *Eedi2020* sub-datasets.

and Prime Factors. Similarly, the relationship *Multiples and Lowest Common Multiple* → *Prime Numbers and Prime Factors* leads to higher peak values for the f_{source} parameter than *Prime Numbers and Prime Factors* → *Multiples and Lowest Common Multiple*. Other studied sub-datasets, such as *(Volume of Non-Prisms/Mental Multiplication and Division)* and *(Volume of Non-Prisms/Mental Addition and Subtraction)*, show no significant differences in the peak values between the two directions of the relationship between the two KCs.

We observe regularities between parameters computed from the two directions of the same relationship. It is worth noting that, most of the time, the peak values of the f_{source} and f_{target} parameters are higher for one of the two directions of the relationship.

KC parameters computed from *Kartable* sub-datasets

Finally, we represent in Figure 6.6 the density plot of the E-PRISM learner models ($e\Delta$) parameters induced by each possible configuration of prerequisite relationships from sub-datasets extracted from the real-world *Kartable* dataset.

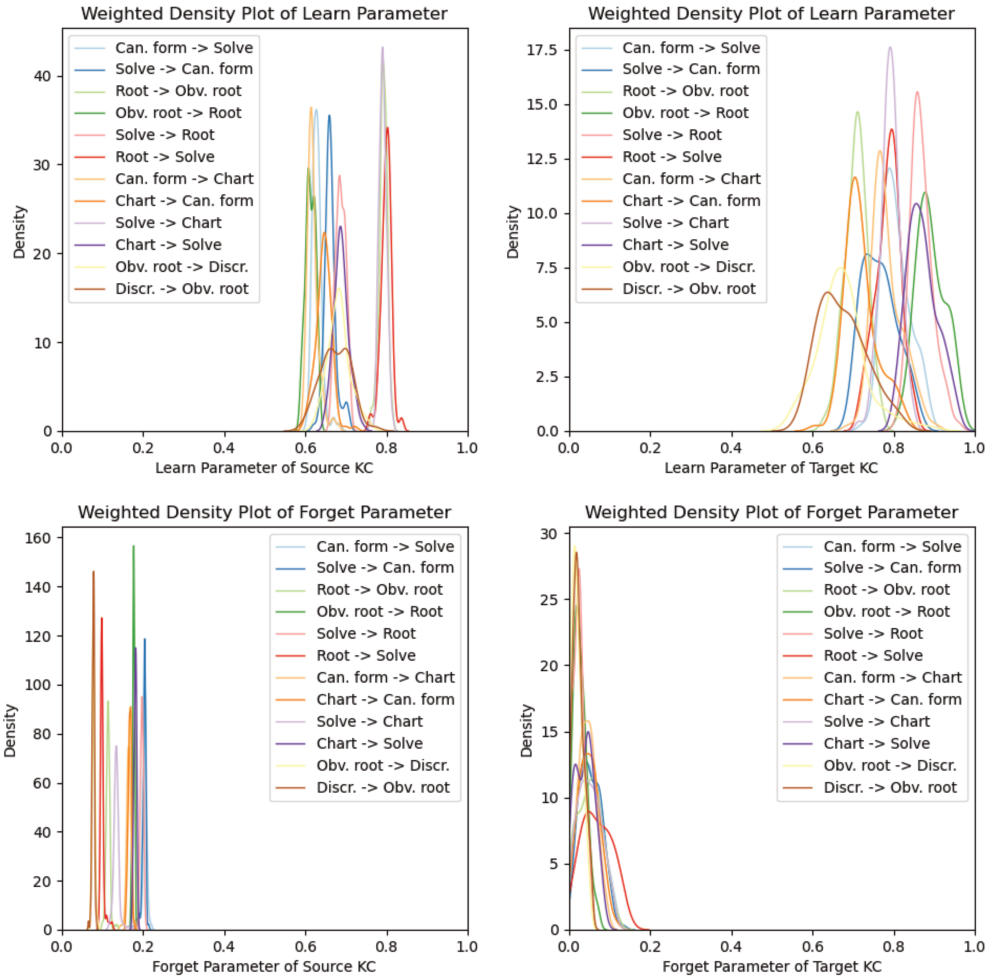


Figure 6.6: Density plots of the learned values of the l and f parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *Kartable* sub-datasets.

First, we notice distinct peaks from source KC learn parameter distribution issued from $e\Delta$s considering *Determine the canonical form of a quadratic polynomial* → *Give the roots of a quadratic polynomial*, *Determine if a real number is a root of a quadratic polynomial* → *Find an obvious root for a quadratic polynomial*, or *Determine if a real number is a root of a quadratic polynomial* → *Give the roots of a quadratic polynomial*. Such as the relationship *Calculate the discriminant of a quadratic polynomial given in expanded form* → *Find an obvious root for a quadratic polynomial*, they also lead to the minimum values for the source KC forget parameter.

We also remark target KC forget parameter has a high density at low values. For instance, the $e\Delta$s that consider *Find an obvious root for a quadratic polynomial* → *Determine if a real number is a root of a quadratic polynomial* and the two possible directions in the sub-dataset *Find an obvious root for a quadratic polynomial / Calculate the discriminant of a quadratic polynomial given in expanded form* have their target KC forget parameter distribution that forms a peak close to zero. On the contrary, the relationship *Determine if a real number is a root of a quadratic polynomial* → *Give the roots of a quadratic polynomial*, which already testifies high values for the source KC learn parameter, shows the highest target KC forget parameter values.

6.3.2 Prerequisite parameters learned from real-world data

The KC parameters have been investigated. We now study the prerequisite parameters' density plots. The prerequisite parameters are the q and s parameters. They will be represented with density plots of their learned values from the E-PRISM learner models considering the different directions of the prerequisite relationship between the two KCs for each sub-dataset. As we did for KC parameters, we aim to identify any distinctions or similarities between the learned parameter distributions and gain insight into the underlying structure of the model. We focus on their ability to capture the relationships between different knowledge components.

Prerequisite parameters computed from *ASSISTments12* sub-datasets

We represent in Figure 6.7 the density plots of the q and s parameter for prerequisite relationships in the *ASSISTments12* sub-datasets.

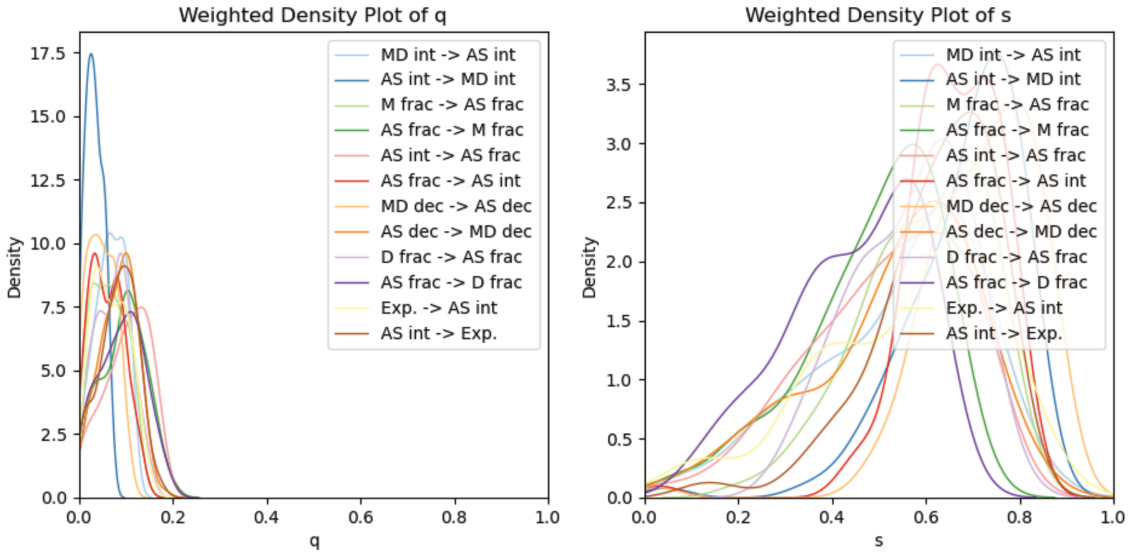


Figure 6.7: Density plots of the learned values of the q and s parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *ASSISTments12* sub-datasets.

As reported on synthetic data, the density plots of the q parameter highly overlap. It is difficult to draw conclusions from these results. Nevertheless, we spot that the density plot of the relationship *Addition and Subtraction Integers* → *Multiplication and Division Integers* is particularly close to zero compared to other plots. This has been reported for E-PRISM models considering the wrong direction of the prerequisite relationship on synthetic data.

On the other hand, the values obtained from learning the s parameter are quite high overall, the plots' peaks being greater than 0.5. Still, we can notice orders between plots from the same sub-datasets. For instance, the peak of the density plot associated with the sub-dataset *(Addition and Subtraction Integers / Addition and Subtraction Fractions)* is significantly lower for the direction *Addition and Subtraction Integers* → *Addition and Subtraction Fractions* than for the direction *Addition and Subtraction Fractions* → *Addition and Subtraction Integers*.

Prerequisite parameters computed from *ASSISTments17* sub-datasets

In Figure 6.8, we represent the density plots of the q and s parameter for prerequisite relationships in the *ASSISTments17* sub-datasets.

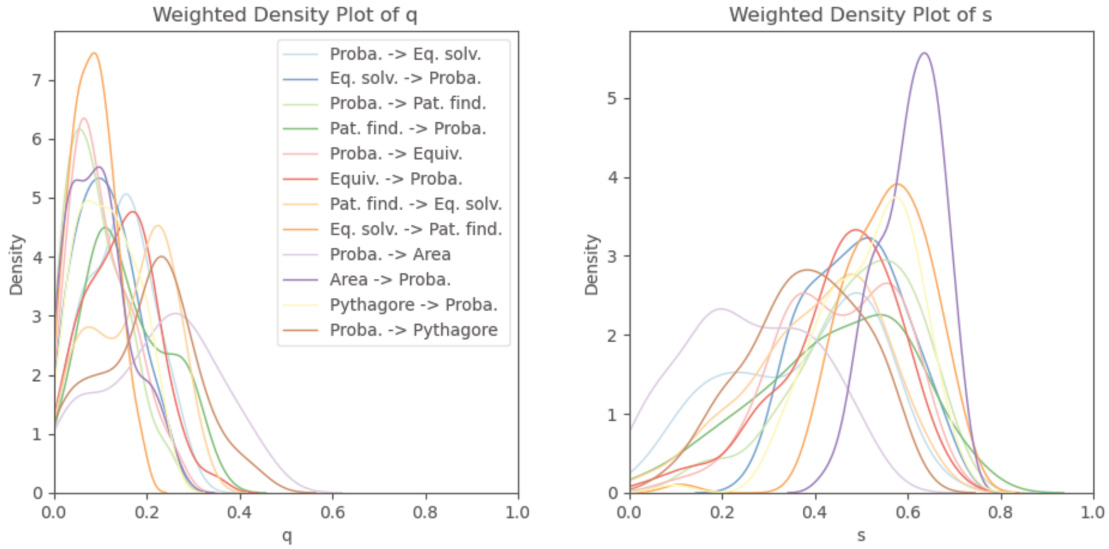


Figure 6.8: Density plots of the learned values of the q and s parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *ASSISTments17* sub-datasets.

Contrary to *ASSISTments12*, the density plots of the q parameter computed on *ASSISTments17* diverge from zero. We observe in Figure 6.8 wider gaps between the two directions of the studied relationships than what we have seen in Figure 6.7. The density plot of the relationship *Probability* → *Area* shows particular high values for q and low values for s. This has been reported for E-PRISM models considering the wrong direction of the prerequisite relationship on synthetic data.

The values obtained for the s parameters are widespread from 0 to 0.8. Nevertheless, except

Probability → *Area*, the other peak values obtained from learning the s parameter are greater than 0.4. Also, all sub-datasets but *(Probability / Pattern finding)* have different values for the two directions of the prerequisite relationship. The sub-dataset *(Probability / Area)* even shows a large gap in the s parameter value between the two possible directions for the relationship.

Prerequisite parameters computed from *Eedi2020* sub-datasets

In Figure 6.9, we represent the density plots of the q and s parameters learned from sub-datasets extracted from the *Eedi2020* dataset.

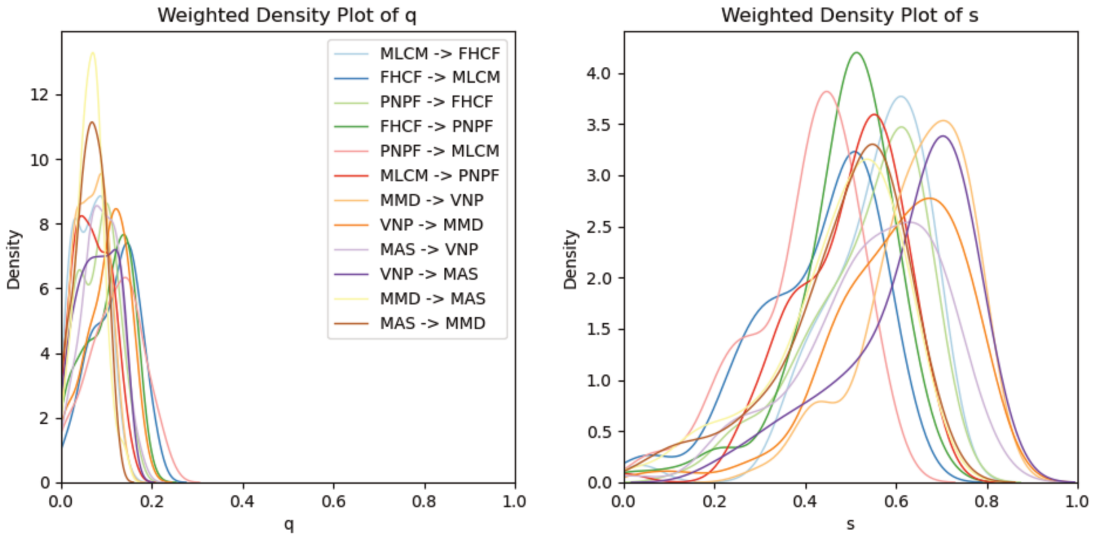


Figure 6.9: Density plots of the learned values of the q and s parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *Eedi2020* sub-datasets.

The results in Figure 6.9 are similar to those in Figure 6.8 about *ASSISTments17* sub-datasets. Indeed, we observe that the q values computed from *Eedi2020* sub-datasets are never too close to zero. This behavior is different from the conduct spotted in synthetic data.

However, similarly to the values computed from *ASSISTments12* data, the values of s are overall quite high, with peaks of distribution being all greater than 0.45. We even do not observe clear and large gaps between the s parameter values computed on the two different directions of the same relationship. If one is able to report an order between distributions, it is not straightforward.

Prerequisite parameters computed from the *Kartable* sub-datasets

Finally, we represent in Figure 6.9 the q and s parameter distribution density plots obtained from learning on *Kartable* sub-datasets.

We first notice that the relationship *Give the roots of a quadratic polynomial* → *Determine if a real number is a root of a quadratic polynomial* relates to a very peaky q parameter distribution,

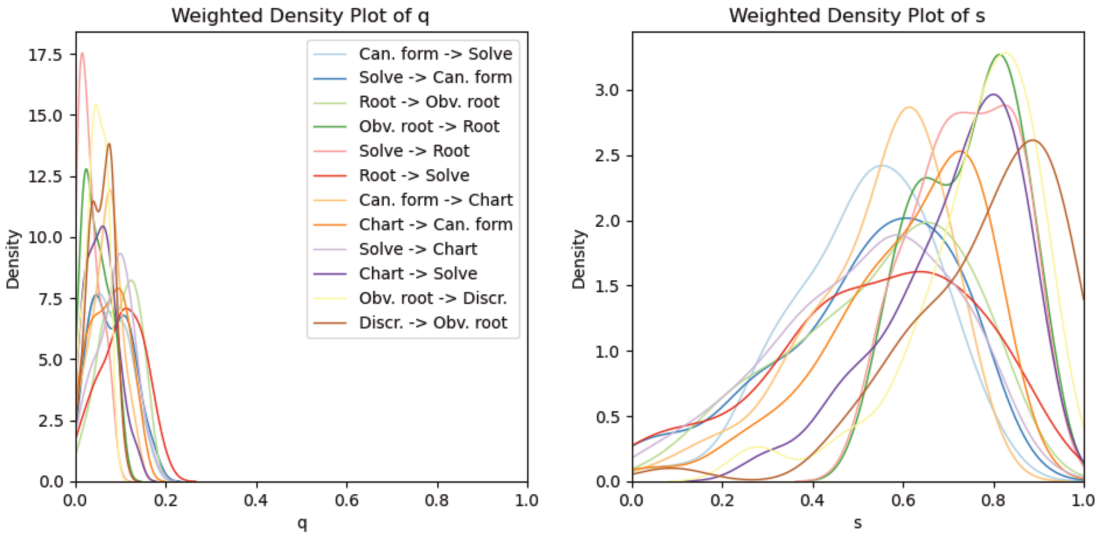


Figure 6.10: Density plots of the learned values of the q and s parameters for an E-PRISM model considering the prerequisite relationships given in the legend. Models are trained on the *Kartable* sub-datasets.

with values being close to zero. On the other hand, the same relationship exhibits large values for the s parameter. These phenomena, already reported for some relationships in the *ASSISTments12* dataset, have been reported for E-PRISM models considering the wrong direction of the prerequisite relationship on synthetic data.

The s values computed from *Kartable* sub-datasets can reach considerably large values. For instance, the relationship *Calculate the discriminant of a quadratic polynomial given in expanded form* $\rightarrow$ *Find an obvious root for a quadratic polynomial* has a peak near 0.9, which is significant for a parameter to which the value should relate to a causal effect. All distribution peaks are greater than 0.5, much higher than those spotted on synthetic data when considering the correct direction.

6.4 Discovering prerequisite relationships between knowledge components from E-PRISM

One of the goals of the E-PRISM learned parameters study is to learn the prerequisite structure of the domain model, particularly by evaluating the performance of the E-PRISM framework in predicting the learner's performance. To this end, we have designed a series of experiments that compare different instances of the E-PRISM learner model under different conditions.

We show that the performance metrics used classically to evaluate EDM algorithms are not well adapted for interpreting the relevance of the learned parameter's values. For this reason, these experiments should assist in defining custom metrics allowing the correct interpretation of the learned parameters.

6.4.1 Defining metrics to measure the existence and the strength of a prerequisite relationship

The previous observations about the learned E-PRISM parameters on synthetic data support the importance of considering the direction of the prerequisite relationships in the E-PRISM framework. The chosen prerequisite direction significantly impacts the accuracy of the parameter learning process, as reported in Section 6.1. We introduce new metrics for estimating the existence, direction, and strength of a prerequisite relationship based on our observations from the study of KC and prerequisite E-PRISM parameters. These metrics are applied when using the E-PRISM framework on real-world datasets to discover the prerequisite structure of the domain model.

Parameter distribution metric

In addition to the $\mathcal{M}_1$ metric, introduced in Section 6.1.4 and based on the classical RMSE measure, we consider a custom metric $\mathcal{M}_2$ defined by rules that compare peak values of the parameter distributions, accordingly to the observations we have made on synthetic data. This metric is expressed in Equation 6.2. It is calculated by comparing the peak values of each parameter value distribution for the E-PRISM learner models considering the correct direction and the wrong direction of the prerequisite relationship. It allows us to determine the direction of the prerequisite relationship by comparing the peak values of the parameter distributions.

$$\begin{aligned}\mathcal{M}_2(\mathfrak{A} \to \mathfrak{B}) = \frac{1}{6}\Big(&\mathbb{1}(l_{\mathfrak{A}}^{\mathfrak{A}\to\mathfrak{B}} > l_{\mathfrak{A}}^{\mathfrak{B}\to\mathfrak{A}}) + \mathbb{1}(l_{\mathfrak{B}}^{\mathfrak{A}\to\mathfrak{B}} < l_{\mathfrak{B}}^{\mathfrak{B}\to\mathfrak{A}}) \\ &+ \mathbb{1}(f_{\mathfrak{A}}^{\mathfrak{A}\to\mathfrak{B}} < f_{\mathfrak{A}}^{\mathfrak{B}\to\mathfrak{A}}) + \mathbb{1}(f_{\mathfrak{B}}^{\mathfrak{A}\to\mathfrak{B}} > f_{\mathfrak{B}}^{\mathfrak{B}\to\mathfrak{A}}) \\ &+ \mathbb{1}(q^{\mathfrak{A}\to\mathfrak{B}} > q^{\mathfrak{B}\to\mathfrak{A}}) + \mathbb{1}(s^{\mathfrak{A}\to\mathfrak{B}} < s^{\mathfrak{B}\to\mathfrak{A}})\Big)\end{aligned} \tag{6.2}$$

where $\mathbb{1}$ is the identity function.

$\mathcal{M}_2$ indicates the prerequisite relationship's existence and direction. It ranges from 0 to 1. The greater $\mathcal{M}_2(\mathfrak{A} \rightarrow \mathfrak{B})$, the most likely the existence of the $\mathfrak{A} \rightarrow \mathfrak{B}$ relationship.

s parameter metric

Based on the previous observations on synthetic data, we propose a second custom metric based on the distribution of the parameter s. We have chosen s as it relates to the probability that the prerequisite is, in practice, unnecessary, as explained in Section 4.2. This second metric $\mathcal{M}_3$ is calculated by determining the proportion of learned values of s lower than 0.2 obtained in the parameter learning runs. It is defined in Equation 6.3.

$$\mathcal{M}_3 = \frac{1}{K} \frac{\text{Number of learned parameter values lower than 0.2}}{\text{Total number of learned parameter values}} \tag{6.3}$$

with K a normalizing constant.

This metric allows us to define the strength of the prerequisite relationship. The closer to 1 the value of $\mathcal{M}_3$, the stronger the prerequisite relationship between the two considered KCs.

By combining these three metrics, we should gain a deeper understanding of the interpretability of the E-PRISM learned parameters and how they can be employed to retrieve the domain model's prerequisite structure (existence, direction, and strength) in E-PRISM.

6.4.2 Study of the proposed metrics

We present the values of the proposed metrics after the training runs on the real-world sub-datasets. We propose to compare their values and extract prerequisite structures from each dataset.

Study of the RMSE-based metric

The RMSE values obtained on the sub-datasets are presented in Appendix C to provide a detailed analysis of the results obtained on real-world data by computing the $\mathcal{M}_1$ metric. We resume these results in Figure 6.11, where we present the gap between the RMSE values obtained on the $e\Delta$s considering the different directions of the prerequisite relationship. The table indicates which direction has the best RMSE value on training data for each sub-dataset. This allows for a quick and easy comparison of the results and provides a clear overview of the performance of the E-PRISM framework on real-world data.

Study of the comparison of parameter distribution metric

We analyze the parameter distribution obtained from the E-PRISM learner models ($e\Delta$) trained on the real-world sub-datasets. We summarize the values of our custom metric $\mathcal{M}_2$ computed on the parameter distributions in Figure 6.12. This allows for an easy comparison of the distribution of parameter values obtained from the $e\Delta$s, considering different directions

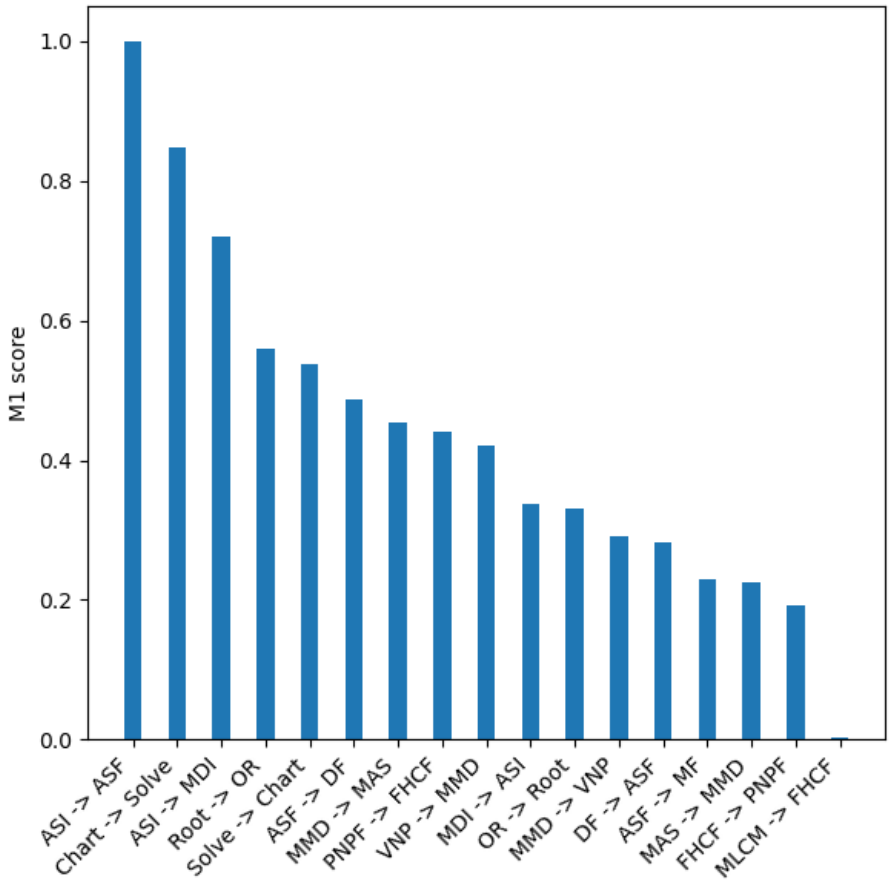


Figure 6.11: $\mathcal{M}_1$ score for every prerequisite relationship that verifies $\mathcal{M}_1 > 0$.

of the prerequisite relationship. This comparison should enable us to draw conclusions on the prerequisite relationship's existence, direction, and strength in real-world data.

Study of the s parameter metric

Finally, we focus on studying the s parameter distribution, which is a key indicator of the prerequisite relationship's existence, direction, and strength. The s parameter distribution for each sub-dataset for both possible directions of the prerequisite relationship is analyzed to compute $\mathcal{M}_3$, based on the proportion of s parameter values lower than 0.2. These values, summarized in Figure 6.13, serve as a quick reference for analyzing the s parameter distribution.

6.4.3 Comparing the metrics

We now compare these metrics against each other to understand the correlation between the masteries of KCs and the existence, direction, and strength of the prerequisite relationships

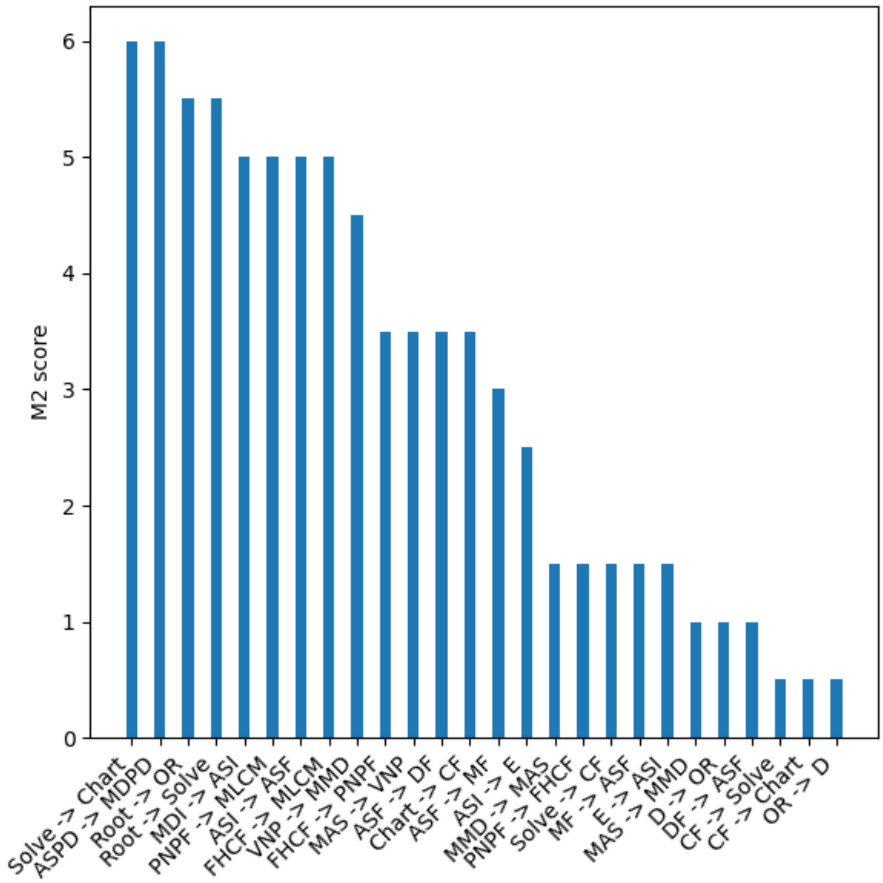


Figure 6.12: $\mathcal{M}_2$ score for every prerequisite relationship that verifies $\mathcal{M}_2 > 0$.

in real-world data.

Relative agreement between metrics

First, we study the relative agreement between introduced metrics for asserting the correctness of the inferred prerequisite structure. To do so, we compute the Cohen kappa [VKS10] between the metric predictors. For each sub-dataset, we evaluate the reliability between metrics on the existence and direction of the corresponding prerequisite relationship.

- For every KCs $\mathfrak{A}$ and $\mathfrak{B}$, we define the predictors on the existence of the prerequisite relationship $\mathfrak{A} \to \mathfrak{B}$ from $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ as follows: $\mathfrak{A} \to \mathfrak{B}$ is predicted to exist according to each metric $\mathcal{M}_i$ if $\mathcal{M}_i(\mathfrak{A} \to \mathfrak{B}) > 0$.
- Similarly, the predictors for the correct direction of the prerequisite relationship from $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ are defined as follows: $\mathfrak{A} \to \mathfrak{B}$ is predicted to exist according to each metric $\mathcal{M}_i$ if $\mathcal{M}_i(\mathfrak{A} \to \mathfrak{B}) > \mathcal{M}_i(\mathfrak{B} \to \mathfrak{A})$.

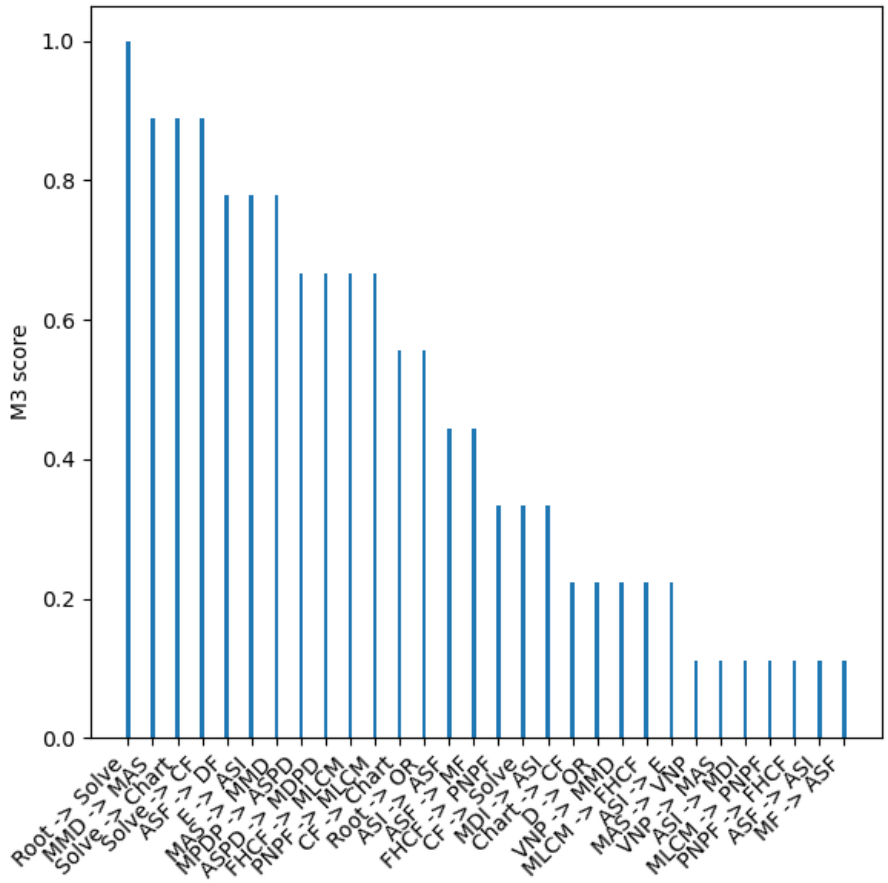


Figure 6.13: $\mathcal{M}_3$ score for every prerequisite relationship that verifies $\mathcal{M}_3 > 0$.

We also introduce a predictor that combines the two conditions, and we present the results in Table 6.13.

		Existence	Direction	Ex. + Dir.
$\mathcal{M}_1$	$\mathcal{M}_2$	0.133	0.325	0.111
$\mathcal{M}_1$	$\mathcal{M}_3$	-0.071	0.55	0.117
$\mathcal{M}_2$	$\mathcal{M}_3$	0.053	0.55	0.778

Table 6.13: Cohen kappa values obtained from measuring the agreement of metrics $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ on the existence and direction of the prerequisite relationships.

We observe that the predictors of the existence of the prerequisite relationship give different results depending on the employed metric. The predictors for the direction of the prerequisite relationships grossly agree with each other, especially $\mathcal{M}_1$ with $\mathcal{M}_3$ and $\mathcal{M}_2$ with $\mathcal{M}_3$. Finally, when considering the two conditions in the predictor, we observe a strong agreement

between $\mathcal{M}_2$ and $\mathcal{M}_3$, $\mathcal{M}_2$ and $\mathcal{M}_3$ then suggest the same relationships to be part of the prerequisite structure of the domain. On the other hand, we observe a weak agreement (near random) between $\mathcal{M}_1$ and the other metrics.

This result suggests that RMSE, even if it can be interpreted as a first filter to determine the existence of the prerequisite structure with $\mathcal{M}_1$, is not sufficient to infer the prerequisite relationships from data. Nevertheless, even if the relevance of $\mathcal{M}_2$ and $\mathcal{M}_3$ have been confirmed by the results, they should be compared with the predictions of experts to assess that the joint agreement between $\mathcal{M}_2$ and $\mathcal{M}_3$ indeed corresponds to the correct prerequisite structure.

Inferred prerequisite relationships

Now we have studied the metrics, and how they relate to the prerequisite structure of the domain model. We focus on the particular prerequisite relationships highlighted by our measures. Namely, we take a further look at the relationships for which the existence of the relationship is predicted accordingly to $\mathcal{M}_2$ and $\mathcal{M}_3$ metrics. We no longer consider the criteria on $\mathcal{M}_1$ to be relevant. We represent in Table 6.14 the relationships that show $\mathcal{M}_2 > 0$ and $\mathcal{M}_3 > 0$. We order the relationships by descending $\mathcal{M}_3$ values.

Some of the relationships enlightened in Table 6.14 are prerequisite relationships according to common knowledge. In particular, relationships between addition KCs and multiplication KCs are greatly represented. Nevertheless, it also suggests that *Multiplication and Division Integers* is a requirement of *Addition and Subtraction Integers*. These relationships should be submitted for the approval of experts in the domain.

Table 6.14 also prints the orderings $\mathcal{O}_2$ and $\mathcal{O}_3$ of the relationship in descending values of $\mathcal{M}_2$ and $\mathcal{M}_3$ for each inferred prerequisite relationship from the $\mathcal{M}_2$ and $\mathcal{M}_3$ predictors. We use these ordering to determine the strength of the prerequisite relationship. Metrics $\mathcal{M}_2$ and $\mathcal{M}_3$ both show great performance for relationships *Determine if a real number is a root of a quadratic polynomial* $\rightarrow$ *Give the roots of a quadratic polynomial*, *Give the roots of a quadratic polynomial* $\rightarrow$ *Give the sign chart of a quadratic polynomial*, and *Addition and Subtraction Positive Decimals* $\rightarrow$ *Multiplication and Division Positive Decimals*. Thanks to the $\mathcal{M}_2$ and $\mathcal{M}_3$ metrics, we can clearly observe that these detected prerequisite relationships are coherent with the mathematics domain knowledge.

Relationship	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{O}_2$	$\mathcal{M}_3$	$\mathcal{O}_3$
Root → Solve	-0.190	0.917	3	1.00	1
Solve → Chart	0.538	1.00	1	0.889	2
MMD → MAS	0.455	0.250	13	0.889	2
Solve → CF	-0.026	0.250	13	0.889	2
ASF → DF	0.486	0.583	10	0.778	5
ASPD → MDPD	-0.255	1.00	1	0.667	6
FHCF → MLCM	-0.025	0.833	5	0.667	6
PNPF → MLCM	-0.039	0.833	5	0.667	6
Root → OR	0.560	0.917	3	0.556	9
ASI → ASF	1.00	0.833	5	0.444	10
ASF → MF	0.230	0.500	12	0.444	10
MDI → ASI	0.338	0.833	5	0.333	12
FHCF → PNPF	0.192	0.583	10	0.333	12
VNP → MMD	0.422	0.750	9	0.222	14
D → OR	-0.334	0.167	15	0.222	14

Table 6.14: Scores of the metrics $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ on relationships that have been predicted as prerequisites according to the $\mathcal{M}_2$ and $\mathcal{M}_3$ predictors.

6.5 Discussion

In conclusion, this chapter has shown that E-PRISM can be a performant framework for predicting the learner's performance. These results are particularly great on synthetic data. Still, we have remarked that state-of-the-art cognitive diagnosis algorithms on real-world data can outperform E-PRISM. We have ended up to the conclusion that having E-PRISM not differentiating observable and latent variables can be problematic, especially when predicting from real-world datasets.

However, the E-PRISM learner model is based on a set of interpretable parameters that sense the causal effect of the learning process and the structure of prerequisite relationships in a specific domain over time. Consequently, it can be used to discover the prerequisite structure expressed in the data. Our study demonstrates the ability of these parameters to compute new metrics, such as $\mathcal{M}_2$ and $\mathcal{M}_3$, which can infer the existence, direction, and strength of prerequisite relationships. These new measures showed relative agreement on the

extracted prerequisite structure that differs from measures from usual metrics such as $\mathcal{M}_1$.

This novel data mining approach of the domain model prerequisite structure to mathematics datasets has indicated the existence of common knowledge prerequisite relationships. However, further research is necessary to verify the effectiveness of these predictions by examining each inferred relationship from an expert's point of view.

Chapter 7

Conclusions and perspectives

In this work, we have introduced the E-PRISM framework, for Embedding Prerequisite Relationships In Student Modeling. We now summarize the findings of our study with E-PRISM, which has revealed to rely on an interpretable and tractable learner model. We have developed this framework as a tool for student modeling, and we have explored the potential applications of E-PRISM.

This conclusion is organized into three sections, each focusing on a specific aspect of E-PRISM and its potential impacts. First, we highlight the strengths of the E-PRISM framework, specifically its learner model. Then, we recall how this work can be seen as a new perspective for complex system modeling using ICI-based Bayesian Networks (BNs). Finally, we explore the potential of E-PRISM for discovering the prerequisite structure of the domain model and how this information could be adapted to the learner's scale.

7.1 E-PRISM, an interpretable and tractable learner model

The main goal of this work was to design and evaluate E-PRISM, a novel framework for student modeling. We have extensively introduced it in Chapter 4.

7.1.1 A learner model that considers the prerequisite structure

E-PRISM is based on dynamic Bayesian networks to infer the learner's knowledge state through knowledge tracing along the learner's answers. It considers the prerequisite relationships between Knowledge Components (KCs) over time in the network thanks to the clause of Independence of Causal Influences (ICI). Therefore, it models both the dynamicity of learning and the causal effect of the prerequisite relationships between knowledge components. We showed that introducing prerequisite relationships to the model increases the performance of E-PRISM to predict the correctness of students' answers to the next question.

7.1.2 A constrained number of parameters

One important aspect is the constraints applied to the model that allow a reduction of the number of parameters. These constraints come from incorporating the ICI-model Conditional

Probability Distributions (CPDs) in the learner model. Using ICI-based Bayesian networks to model the causal effects of both the learning process dynamics and the domain model prerequisite structure has drastically reduced the number of parameters compared with similar learner models. We may introduce additional assumptions on the model parameter values in the future. For instance, we can imagine including forbidden values or supplementary equalities between parameters extracted from experts' knowledge.

7.1.3 Enhanced interpretability

The low number of parameters has enhanced the interpretability of the E-PRISM learner model. They are attached to larger features which are more understandable. Interpretability is indeed achieved when the length of conditions for describing the output is small. For instance, in other kinds of interpretable machine learning models, shorter trees lead to more interpretable decision trees, and a small number of rules implies more interpretable rule-based estimators [MBGW21]. While other BN approaches for prerequisite relationships use tabular CPDs to depict the prerequisite structure involving a knowledge component $\mathfrak{X}$, each $\mathfrak{X}$'s prerequisite is attached to two parameters in the E-PRISM learner model. The model parameters are easier to understand and interpret.

This constraint is a key characteristic that sets E-PRISM apart from other learner system models. It makes it a promising tool for learner modeling, especially when considering the prerequisite structure of the domain knowledge. E-PRISM parameters give insightful easy-to-interpret information on the prerequisite structure of the domain knowledge.

7.1.4 A promising tool for discovering prerequisite relationships

The E-PRISM framework is not only a tool for providing a learner model to various ITS. It also gives a glimpse into the prerequisite structure of the domain model depicted in the data and opens the path for adapting it to individuals. E-PRISM parameters indeed sense the causal effect of the learning process and the structure of prerequisite relationships in a specific domain over time. Our study demonstrates the possibility of computing metrics, such as $\mathcal{M}_2$ and $\mathcal{M}_3$, from these parameters. We can infer the existence, direction, and strength of prerequisite relationships from these new measures. They have proved they are promising alternatives to usual metrics, studied through $\mathcal{M}_1$. Our results, applied to the domain of mathematics, indicate the existence of common knowledge prerequisite relationships. However, further research is necessary to verify the effectiveness of these predictions by examining each inferred relationship from an expert's point of view.

7.2 A new perspective for complex system modeling

Even if our work is under the educational spectrum, some of our results could be extended to more general complex system modeling. In particular, the use of ICI-based Bayesian

networks and the techniques we have implemented in them.

7.2.1 ICI-based Bayesian networks

We have proposed a new architecture for BNs based on the clause of independence of causal influences. This architecture is particularly adapted when modeling complex systems with scarce data. It reduces the number of parameters in the network and allows for better interpretability of the model. One of the perspectives of this work would be to generalize the kinds of complex systems that can be modeled with E-PRISM. Having an interpretable model for various complex systems could be an exciting outlook because it would allow the user to understand the underlying relationships between the variables and how they influence each other, thanks to easy-to-interpret parameters.

7.2.2 A novel approach for approximate inference in ICI-based BNs

Furthermore, we have presented a new approach for approximate inference in large Bayesian networks with deterministic relationships. This new technique addresses the convergence issue in traditional techniques, such as Gibbs sampling, encountered with ICI-based BNs. Our approach is based on Blocking Gibbs sampling and allows the convergence of the inference for reducible Markov processes. We have extensively evaluated our approach through a series of experiments. We have shown that it converges significantly better than Gibbs sampling, a widely used method for approximate inference in BNs. We have shown that our approach can make accurate predictions and obtain reliable results, even when data is extremely scarce.

7.2.3 A performant parameter learning algorithm in ICI-based BNs

More generally, these results imply the opportunity for parameter learning in ICI-based BNs, which can be challenging in the context of scarce data. We have introduced the Monte-Carlo Expectation Maximization (MCEM) algorithm mobilizing our custom Blocking Gibbs sampling to perform the E-step. We have shown the convergence of our parameter learning algorithm.

Because parameter learning is the foundation of structure learning for Bayesian networks, this algorithm could be applied to perform structure learning in future work. Nevertheless, parameter learning is computationally costly. Therefore, extensive work on optimizing the parameter learning algorithm should be done first. In particular, using high-level programming language instead of `Python` may reduce the computational limitations spotted in this work.

7.3 Future research with the E-PRISM framework

Although this study has provided valuable insights into student modeling, and more particularly on using the domain knowledge prerequisite structure, there are still many unanswered

questions and areas that require further exploration. As such, we outline potential avenues for future research, with the aim of building on the findings of this study and contributing to the broader field of educational data mining.

7.3.1 Differentiating latent and observable variables

One of our work's pain points is that we made no difference between the observable and latent variables in the E-PRISM learner model. Usually, Bayesian network approaches for modeling students, such as BKT, consider an observable variable related to one or multiple latent variables. In E-PRISM, we have supposed that the KC mastery variables are directly observed with learner's transactions because of computational limitations. This report leads to the first perspective for this work. Such as other Bayesian approaches for learner modeling, the slip and guess phenomena should be integrated into the E-PRISM learner model. In particular, it would allow E-PRISM to be less prone to overfitting. Note that the foundations of this research perspective have already been implemented in the package attached to the E-PRISM framework.

7.3.2 Blowing computational limitations up

We highlighted the numerous sources of computational complexity that arise from the iterative approximate inference technique used in the E-PRISM framework. The computational complexity of this technique depends on the number of knowledge components, learner transactions, and Gibbs iterations. Additionally, multiple initializations are required for parameter learning. These limitations pose significant challenges for the analysis of large and complex educational datasets. To address these challenges, future research should focus on implementing deeper parallelization and code optimization. We have already begun to address the challenge of multiple initializations required for parameter learning by implementing parallelization techniques. However, there is still room for further optimization and refinement of these techniques to achieve even greater speed and efficiency.

By developing solutions to these computational challenges, we can advance the capacity of E-PRISM to retrieve the prerequisite structure of the domain knowledge by performing structure learning and, then, considering more knowledge components. This, in turn, can help to inform the development of more effective educational interventions and improve student learning outcomes.

7.3.3 Analyzing the results with experts

Another potential perspective for future research is to seek expert analysis of the results, particularly because of the absence of predictors for prerequisite relationships. There are several constraints to this approach. One major challenge is the availability of experts, as it can be difficult to find individuals with the necessary expertise and time to devote to such an analysis. Additionally, achieving mutual agreement among experts can be challenging,

as it requires consistent feedback and discussion to arrive at a shared understanding of the findings. Finally, achieving statistical significance for rigorous analysis can also be a challenge, as it may require input from a large number of experts.

Nonetheless, despite these challenges, obtaining experts' assessment of the outcomes could provide valuable insights into the correctness of the extracted prerequisite structure from the E-PRISM learner model. Moreover, this could enhance our insights into the E-PRISM framework. By comparing the expert mutual agreement with the E-PRISM parameter values, we can gain a more comprehensive understanding of the model's interpretability and validity.

7.3.4 Restricting model learning to sub-populations

This work presents an approach for inferring prerequisite relationships in educational data mining by analyzing the parameters of an interpretable learner model. Therefore, by reducing training datasets to well-chosen sets of learners, we can imagine that the discovered prerequisite structure would be adapted to the corresponding set of learners. We can then propose the perspective of having the discovery of prerequisite relationships included in a more extensive process. We could first cluster learners with well-chosen features [MEMB18] and discover the whole prerequisite structure of domain knowledge corresponding to each cluster thanks to multiple instances of E-PRISM parameter learning on the different restricted datasets.

References

[ACKP95] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *The journal of the learning sciences*, 4(2):167–207, 1995.

[AGP91] Yali Amit, Ulf Grenander, and Mauro Piccioni. Structural image restoration through deformable templates. *Journal of the American Statistical Association*, 86(414):376–387, 1991.

[AKIB19] Abir Abyaa, Mohammed Khalidi Idrissi, and Samir Bennani. Learner modelling: systematic review of the literature from the last 5 years. *Educational Technology Research and Development*, 67(5):1105–1143, 2019.

[AMRK06] Vincent Aleven, Bruce Mclaren, Ido Roll, and Kenneth Koedinger. Toward meta-cognitive tutoring: A model of help seeking with a cognitive tutor. *International Journal of Artificial Intelligence in Education*, 16(2):101–128, 2006.

[AMSK09] Vincent Aleven, Bruce M Mclaren, Jonathan Sewall, and Kenneth R Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, 19(2):105–154, 2009.

[AT19] Ashraf Mohamed Abou Tabl. *Big Data Analytics for Complex Systems*. PhD thesis, University of Windsor (Canada), 2019.

[AYGA15] Maha Al-Yahya, Remya George, and Auhood Alfaries. Ontologies in e-learning: review of the literature. *International Journal of software engineering and its applications*, 9(2):67–84, 2015.

[BB01] Pierre Baldi and Søren Brunak. *Bioinformatics: the machine learning approach*. MIT press, 2001.

[BL04] Ronald Brachman and Hector Levesque. *Knowledge representation and reasoning*. Elsevier, 2004.

[BM07] Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. In *The adaptive web*, pages 3–53. Springer, 2007.

[Bur82] Richard R Burton. Diagnosing bugs in a simple procedural skill. *Intelligent Tutoring Systems*, pages 157–184, 1982.

[BWP21] Anirudhan Badrinath, Frederic Wang, and Zachary Pardos. pybkt: an accessible python library of bayesian knowledge tracing models. *Proceedings of The 14th International Conference on Educational Data Mining*, pages 468–474, 2021.

[CA94] Albert T Corbett and John R Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[Can01] Ward Canfield. Aleks: A web-based intelligent tutoring system. *Mathematics and Computer Education*, 35(2):152, 2001.

[CCD95] Gilles Celeux, Didier Chauveau, and Jean Diebolt. *On stochastic versions of the EM algorithm*. PhD thesis, INRIA, 1995.

[CD88] Gilles Celeux and Jean Diebolt. *A random imputation principle: the stochastic EM algorithm*. PhD thesis, INRIA, 1988.

[CG77] Brian Carr and Ira P Goldstein. Overlays: A theory of modelling for computer aided instruction. Technical report, Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1977.

[CG92] George Casella and Edward I George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[CG95] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[CGV02] Cristina Conati, Abigail Gertner, and Kurt Vanlehn. Using bayesian networks to manage uncertainty in student modeling. *User modeling and user-adapted interaction*, 12(4):371–417, 2002.

[CKJ06] Hao Cen, Kenneth Koedinger, and Brian Junker. Learning factors analysis–a general method for cognitive model evaluation and improvement. In *International conference on intelligent tutoring systems*, pages 164–175. Springer, 2006.

[CKP+06] Hyun Jin Cha, Yong Se Kim, Seon Hee Park, Tae Bok Yoon, Young Mo Jung, and Jee-Hyong Lee. Learning styles diagnosis based on user interface behaviors for the customization of learning interfaces in an intelligent tutoring system. In *International Conference on Intelligent Tutoring Systems*, pages 513–524. Springer, 2006.

[CM21] Fabio Gagliardi Cozman and Hugo Neri Munhoz. Some thoughts on knowledge-enhanced machine learning. *International Journal of Approximate Reasoning*, 136:308–324, 2021.

[CMPdlC+05] Cristina Carmona, Eva Millán, José-Luis Pérez-de-la Cruz, Mónica Trella, and Ricardo Conejo. Introducing prerequisite relations in a multi-layered bayesian student model. In *International conference on user modeling*, pages 347–356. Springer, 2005.

[Coo90] Gregory F Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

[Cow98] Robert Cowell. Advanced inference in bayesian networks. In *Learning in graphical models*, pages 27–49. Springer, 1998.

[CPBV19] Benoît Choffin, Fabrice Popineau, Yolaine Bourda, and Jill-Jênn Vie. Das3h: modeling student learning and forgetting for optimally scheduling distributed practice of skills. *Proceedings of The 12th International Conference on Educational Data Mining*, pages 29–38, 2019.

[CV13] Konstantina Chrysafiadi and Maria Virvou. Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications*, 40(11):4715–4729, 2013.

[CWL15] Yang Chen, Pierre-Henri Wuillemin, and Jean-Marc Labat. Discovering prerequisite structure of skills through probabilistic association rules mining. In *Proceedings of The 8th International Conference on Educational Data Mining*, pages 117–124, 2015.

[DA15] Melissa C Duffy and Roger Azevedo. Motivation matters: Interactions between achievement goals and agent scaffolding for self-regulated learning within an intelligent tutoring system. *Computers in Human Behavior*, 52:338–348, 2015.

[dBCA08] Ryan SJ d Baker, Albert T Corbett, and Vincent Aleven. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *International conference on intelligent tutoring systems*, pages 406–415. Springer, 2008.

[DBH18] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 0210–0215. IEEE, 2018.

[DD06] F Javier Dıez and Marek J Druzdzel. Canonical probabilistic models for knowledge engineering. *UNED, Madrid, Spain, Technical Report CISIAD-06-01*, 2006.

[DdB12] Michel C Desmarais and Ryan SJ d Baker. A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction*, 22(1):9–38, 2012.

[DF12] Jean-Paul Doignon and Jean-Claude Falmagne. *Knowledge spaces*. Springer Science & Business Media, 2012.

[DGW20] Gaspard Ducamp, Christophe Gonzales, and Pierre-Henri Wuillemin. aGrUM/pyAgrum : a Toolbox to Build Models and Algorithms for Probabilistic Graphical Models in Python. In *10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 609–612, Skørping, Denmark, September 2020.

[Dic] Cambridge Dictionary. Machine learning | english meaning - cambridge dictionary. https://dictionary.cambridge.org/dictionary/english/machine-learning. Accessed: 2022-01-13.

[DL21] Xinyi Ding and Eric C Larson. On the interpretability of deep learning based models for knowledge tracing. *Artificial Intelligence in Education. 2020 Jun 10; 12164*, pages 185–190, 2021.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[DMG06] Michel C Desmarais, Peyman Meshkinfam, and Michel Gagnon. Learned student models with item to item knowledge structures. *User Modeling and User-Adapted Interaction*, 16:403–434, 2006.

[DRK08] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Probabilistic inductive logic programming*, pages 1–27. Springer, 2008.

[DSA11] Rónán Daly, Qiang Shen, and Stuart Aitken. Learning bayesian networks: approaches and issues. *The knowledge engineering review*, 26(2):99–157, 2011.

[edt] Edtech market analysis. https://www.globaldata.com/store/report/edtech-market-analysis/. Accessed: 2022-01-14.

[FH06] Mingyu Feng and Neil T Heffernan. Informing teachers live about student learning: Reporting in the assistment system. *Technology Instruction Cognition and Learning*, 3(1/2):63, 2006.

[FHK09] Mingyu Feng, Neil Heffernan, and Kenneth Koedinger. Addressing the assessment challenge with an online system that tutors as it assesses. *User modeling and user-adapted interaction*, 19(3):243–266, 2009.

[FRHG19] Ying Fang, Zhihong Ren, Xiangen Hu, and Arthur C Graesser. A meta-analysis of the effectiveness of aleks on learning. *Educational Psychology*, 39(10):1278–1292, 2019.

[FVNN10] Philippe Fournier-Viger, Roger Nkambou, and Engelbert Mephu Nguifo. Building intelligent tutoring systems for ill-defined domains. *Advances in intelligent tutoring systems*, pages 81–101, 2010.

[GBL06] Carolina González, Juan C Burguillo, and Martín Llamas. A qualitative comparison of techniques for student modeling in intelligent tutoring systems. In *Proceedings. Frontiers in Education. 36th Annual Conference*, pages 13–18. IEEE, 2006.

[GFK+07] Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Benjamin Taskar. Probabilistic relational models. *Introduction to statistical relational learning*, 8, 2007.

[GG84] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[GKS+20] Theophile Gervet, Ken Koedinger, Jeff Schneider, Tom Mitchell, et al. When is deep learning the best approach to knowledge tracing? *Journal of Educational Data Mining*, 12(3):31–54, 2020.

[GRS95] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[Hae89] Edward H Haertel. Using restricted latent class models to map the skill structure of achievement items. *Journal of Educational Measurement*, 26(4):301–321, 1989.

[HAH+13] Jürgen Heller, Thomas Augustin, Cord Hockemeyer, Luca Stefanutti, and Dietrich Albert. Recent developments in competence-based knowledge space theory. *Knowledge spaces: Applications in education*, pages 243–286, 2013.

[HB96] David Heckerman and John S Breese. Causal independence for probability assessment and inference using bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 26(6):826–831, 1996.

[HBC+21] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto

Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

[Hen88] Max Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *Machine intelligence and pattern recognition*, volume 5, pages 149–163. Elsevier, 1988.

[HH14] Neil T Heffernan and Cristina Lindquist Heffernan. The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, 24(4):470–497, 2014.

[Hry90] Tomas Hrycej. Gibbs sampling in bayesian networks. *Artificial Intelligence*, 46(3):351–363, 1990.

[HSHA06] Jürgen Heller, Christina Steiner, Cord Hockemeyer, and Dietrich Albert. Competence-based knowledge structures for personalised learning. *International Journal on E-learning*, 5(1):75–88, 2006.

[HSR91] Ronald K Hambleton, Hariharan Swaminathan, and H Jane Rogers. *Fundamentals of item response theory*, volume 2. Sage, 1991.

[IKU08] Takashi Isozaki, Noriji Kato, and Maomi Ueno. Minimum free energies with" data temperature" for parameter learning of bayesian networks. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 371–378. IEEE, 2008.

[Jae01] Manfred Jaeger. Complex probabilistic modeling with recursive relational bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32(1):179–220, 2001.

[Jae22] Manfred Jaeger. Learning and Reasoning with Graph Data: Neural and Statistical-Relational Approaches. In *International Research School in Artificial Intelligence in Bergen (AIB 2022)*, volume 99 of *Open Access Series in Informatics (OASIcs)*, pages 5:1–5:42, 2022.

[JCK09] Philip I. Pavlik Jr., Hao Cen, and Kenneth R. Koedinger. Learning factors transfer analysis: Using learning curve analysis to automatically generate domain models. pages 121–130, 2009.

[JJG12] Z Jeremić, J Jovanović, and D Gašević. Student modeling and assessment in intelligent tutoring of software patterns. *Expert Systems with Applications*, 39(1):210–222, 2012.

[JKK95] Claus S Jensen, Uffe Kjærulff, and Augustine Kong. Blocking gibbs sampling in very large probabilistic expert systems. *International Journal of Human-Computer Studies*, 42(6):647–666, 1995.

[JS01] Brian W Junker and Klaas Sijtsma. Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. *Applied Psychological Measurement*, 25(3):258–272, 2001.

[KAH+97] Kenneth R Koedinger, John R Anderson, William H Hadley, Mary A Mark, et al. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1):30–43, 1997.

[KCP12] Kenneth R Koedinger, Albert T Corbett, and Charles Perfetti. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science*, 36(5):757–798, 2012.

[KD90] Mathieu Koppen and Jean-Paul Doignon. How to build a knowledge space by querying an expert. *Journal of Mathematical Psychology*, 34(3):311–331, 1990.

[KDG+02] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[KKSG14] Tanja Käser, Severin Klingler, Alexander Gerhard Schwing, and Markus Gross. Beyond knowledge tracing: Modeling skill topologies with bayesian networks. In *International conference on intelligent tutoring systems*, pages 188–198. Springer, 2014.

[KKSG17] Tanja Käser, Severin Klingler, Alexander G Schwing, and Markus Gross. Dynamic bayesian networks for student modeling. *IEEE Transactions on Learning Technologies*, 10(4):450–462, 2017.

[KLM16] Mohammad Khajah, Robert V Lindsey, and Michael C Mozer. How deep is knowledge tracing? *Proceedings of the 9th International Conference on Educational Data Mining*, pages 94–101, 2016.

[Kol13] A.N. Kolmogorov. *Foundations of the Theory of Probability*. Martino Fine Books, 2013.

[KS89] Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer, 1989.

[Li09] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[LLB18] Hugues Labarthe, Vanda Luengo, and François Bouchet. Analyse de l'hybridation entre les communautés lak, edm et aied. In *PFIA 2018: Journée IA pour l'éducation*, 2018.

[LSPM14] Robert V Lindsey, Jeffery D Shroyer, Harold Pashler, and Michael C Mozer. Improving students' long-term knowledge retention through personalized review. *Psychological science*, 25(3):639–647, 2014.

[Mar99] Brent Martin. Constraint-based modelling: Representing student knowledge. *New Zealand Journal of Computing*, 7(2):30–38, 1999.

[MBGW21] Vincent Margot, Jean-Patrick Baudry, Frederic Guilloux, and Olivier Wintenberger. Consistent regression using data-dependent coverings. *Electronic Journal of Statistics*, 15(1):1743 – 1782, 2021.

[MBL21] Bojan Mihaljević, Concha Bielza, and Pedro Larrañaga. Bayesian networks for interpretable machine learning and optimization. *Neurocomputing*, 456:648–665, 2021.

[MEMB18] Shirin Mojarad, Alfred Essa, Shahin Mojarad, and Ryan S Baker. Data-driven learner profiling based on clustering student behaviors: learning consistency, pace and effort. In *International Conference on Intelligent Tutoring Systems*, pages 130–139. Springer, 2018.

[MG14] Sonia Mandin and Nathalie Guin. Basing learner modelling on an ontology of knowledge and skills. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*, pages 321–323. IEEE, 2014.

[Min01] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369, 2001.

[Mit12] Antonija Mitrovic. Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User modeling and user-adapted interaction*, 22(1):39–72, 2012.

[MJ99] Anders L Madsen and Finn V Jensen. Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1-2):203–245, 1999.

[MLPDLC10] Eva Millán, Tomasz Loboda, and Jose Luis Pérez-De-La-Cruz. Bayesian networks for student model engineering. *Computers & Education*, 55(4):1663–1683, 2010.

[MM00] Michael Mayo and Antonija Mitrovic. Using a probabilistic student model to control problem difficulty. In *International conference on intelligent tutoring systems*, pages 524–533. Springer, 2000.

[Mol22] Christoph Molnar. Interpretable machine learning, 2022.

[Mug91] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

[MWJ99] Kevin Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of the 15th Conf. on Uncertainty in Artificial Intelligence, 1999*, pages 467–475, 1999.

[ND08] Loc Nguyen and Phung Do. Learner model in adaptive learning. *World Academy of Science, Engineering and Technology*, 45(70):395–400, 2008.

[NMB10] Roger Nkambou, Riichiro Mizoguchi, and Jacqueline Bourdeau. *Advances in intelligent tutoring systems*, volume 308. Springer Science & Business Media, 2010.

[Ohl94] Stellan Ohlsson. Constraint-based student modeling. In *Student modelling: the key to individualized knowledge-based instruction*, pages 167–189. Springer, 1994.

[Paq22] Gilbert Paquette. Les fondements de l'ingénierie des environnements numériques d'apprentissage. *Apprendre et enseigner sur le Web: quelle ingénierie pédagogique?*, page 101, 2022.

[PBH+15] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas J Guibas, and Jascha Sohl-Dickstein. Deep knowledge tracing. *Advances in neural information processing systems*, 28, 2015.

[PCK09] Philip I Pavlik, Hao Cen, and Kenneth R Koedinger. Performance factors analysis–a new alternative to knowledge tracing. In *Artificial Intelligence in Education*, pages 531–538. IOS Press, 2009.

[PCWK08] Philip I. Pavlik, Hao Cen, Lili Wu, and Kenneth R. Koedinger. Using item-type performance covariance to improve the skill model of an existing tutor. In *Proceedings of The 1st International Conference on Educational Data Mining*, pages 77–86, 2008.

[PD+18] Zachary A Pardos, Anant Dadu, et al. dafm: Fusing psychometric and connectionist modeling for q-matrix refinement. *Journal of Educational Data Mining*, 10(2):1–27, 2018.

[Pea88] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[Pel15] Radek Pelánek. Metrics for evaluation of student models. *Journal of Educational Data Mining*, 7(2):1–19, 2015.

[Pel17] Radek Pelánek. Bayesian knowledge tracing, logistic models, and beyond: an overview of learner modeling techniques. *User Modeling and User-Adapted Interaction*, 27(3):313–350, 2017.

[Pel20] Radek Pelánek. Managing items and knowledge components: domain modeling in practice. *Educational Technology Research and Development*, 68(1):529–550, 2020.

[PH10] Zachary A Pardos and Neil T Heffernan. Modeling individualization in a bayesian networks implementation of knowledge tracing. In *International conference on user modeling, adaptation, and personalization*, pages 255–266. Springer, 2010.

[PH11] Zachary A Pardos and Neil T Heffernan. Kt-idem: Introducing item difficulty to the knowledge tracing model. In *International conference on user modeling, adaptation, and personalization*, pages 243–254. Springer, 2011.

[PK19] Shalini Pandey and George Karypis. A self-attentive model for knowledge tracing. In *Proceedings of The 12th International Conference on Educational Data Mining*, pages 384–389. International Educational Data Mining Society, 2019.

[PM15] DJ Patil and Hilary Mason. *Data Driven.* " O'Reilly Media, Inc.", 2015.

[PY13] Zachary A Pardos and Michael V Yudelson. Towards moment of learning accuracy. In *AIED 2013 Workshops Proceedings Volume*, volume 4, page 3. Citeseer, 2013.

[QQL+11] Yumeng Qiu, Yingmei Qi, Hanyuan Lu, Zachary A Pardos, and Neil T Heffernan. Does time matter? modeling the effect of time with bayesian knowledge tracing. In *EDM*, pages 139–148, 2011.

[RAKC07] Steven Ritter, John R Anderson, Kenneth R Koedinger, and Albert Corbett. Cognitive tutor: Applied research in mathematics education. *Psychonomic bulletin & review*, 14(2):249–255, 2007.

[Rat96] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of Conference on empirical methods in natural language processing*, pages 133–142, 1996.

[RD06] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1):107–136, 2006.

[Rec09] Mark D Reckase. Multidimensional item response theory models. In *Multidimensional item response theory*, pages 79–112. Springer, 2009.

[Ric79] Elaine Rich. User modeling via stereotypes. *Cognitive science*, 3(4):329–354, 1979.

[SCS00] Jan Maarten Schraagen, Susan F Chipman, and Valerie L Shalin. *Cognitive task analysis*. Psychology Press, 2000.

[She00] Nuala A Sheehan. On the application of markov chain monte carlo methods to genetic analyses on complex pedigrees. *International Statistical Review*, 68(1):83–110, 2000.

[SHPU12] Alexander G. Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction with latent variables for general graphical models. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, ICML'12, page 1659–1666, 2012.

[SP89] R Shachter and Mark Peot. Evidential reasoning using likelihood weighting. In *Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 18–20, 1989.

[SSG14] Richard Scheines, Elizabeth Silver, and Ilya M Goldin. Discovering prerequisite relationships among knowledge components. In *Proceedings of The 7th International Conference on Educational Data Mining*, pages 355–356, 2014.

[SWHM22] Robin Schmucker, Jingbo Wang, Shijia Hu, and Tom M Mitchell. Assessing the performance of online students–new data, new approaches, improved accuracy. *Journal of Educational Data Mining*, 14(1):1–45, 2022.

[TH06] Jonathan L Templin and Robert A Henson. Measurement of psychological disorders using cognitive diagnosis models. *Psychological methods*, 11(3):287, 2006.

[VKS10] Susana M Vieira, Uzay Kaymak, and João MC Sousa. Cohen's kappa coefficient as a performance measure for feature selection. In *International conference on fuzzy systems*, pages 1–8. IEEE, 2010.

[VP98] Angel de Vicente and Helen Pain. Motivation diagnosis in intelligent tutoring systems. In *International Conference on Intelligent Tutoring Systems*, pages 86–95. Springer, 1998.

[Wen86] Etienne Wenger. Artificial intelligence and tutoring systems: Computational approaches to the communication of knowledge. Technical report, Univ. of California, Irvine, 1986.

[WJ+08] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008.

[WLS+21] Zichao Wang, Angus Lamb, Evgeny Saveliev, Pashmina Cameron, Yordan Zaykov, José Miguel Hernández-Lobato, Richard E Turner, Richard G Baraniuk, Craig Barton, Simon Peyton Jones, et al. Instructions and guide for diagnostic questions: The neurips 2020 education challenge. *Proceedings of Machine Learning Research 133:191–205, 2021 NeurIPS 2020 Competition and Demonstration Track*, 2021.

[Woo10] Beverly Park Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning.* Morgan Kaufmann, 2010.

[WPB01] Geoffrey I Webb, Michael J Pazzani, and Daniel Billsus. Machine learning for user modeling. *User modeling and user-adapted interaction*, 11(1):19–29, 2001.

[WT90] Greg CG Wei and Martin A Tanner. A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American statistical Association*, 85(411):699–704, 1990.

[Wu83] CF Jeff Wu. On the convergence properties of the em algorithm. *The Annals of statistics*, pages 95–103, 1983.

[YKG13] Michael V Yudelson, Kenneth R Koedinger, and Geoffrey J Gordon. Individualized bayesian knowledge tracing models. In *International conference on artificial intelligence in education*, pages 171–180. Springer, 2013.

[Yor92] Jeremy York. Use of the gibbs sampler in expert systems. *Artificial Intelligence*, 56(1):115–130, 1992.

[ZBKM18] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[ZVD06] Adam Zagorecki, Mark Voortman, and Marek J Druzdzel. Decomposing local probability distributions in bayesian networks for improved inference and parameter learning. In *FLAIRS Conference*, pages 860–865, 2006.

Appendix A

Details on the Gibbs sampling procedure

We recall that in a Bayesian network (dynamic or not), the Markov blanket of a variable x_i is composed of its parent nodes in the network, its child nodes in the network, and the other parent nodes of its child nodes.

A.1 Initial Bayesian network

In the first place, we focus on variables that belong to the initial Bayesian network $\mathcal{B}_0$. In $\mathcal{B}_0$, for a given KC $\mathfrak{X}$, there are no "transition" nodes $T_{\mathfrak{X}}$ but only the mastery node X^0 and the prerequisite nodes $Z^0_{\mathfrak{X},i}$.

A.1.1 Mastery nodes in $\mathcal{B}_0$

The mastery node only depends on the prerequisite nodes relating the causal effect of its parents on the KC $\mathfrak{X}$: the expression of the CPD that rules the variable X^0 then depends on the prerequisite relationships involving $\mathfrak{X}$.

When $\mathfrak{X}$ is not involved in any prerequisite relationship, the Markov blanket of X^0 is only composed of the variable $T^1_{\mathfrak{X}}$. The CPD that rules X^0 is then expressed in Equation A.1.

$$P(X^0 = +x \mid T^1_{\mathfrak{X}}) = \left(1 + \frac{P(\neg x)P(T^1_{\mathfrak{X}} \mid \neg x)}{P(+x)P(T^1_{\mathfrak{X}} \mid +x)}\right)^{-1} \tag{A.1}$$

The resulting CPD is represented in Table A.1.

When $\mathfrak{X}$ is a root in the prerequisite structure (i.e. if it has no prerequisite knowledge component), the Markov blanket of X^0 is composed of $T^1_{\mathfrak{X}}$ and the variables between the mastery of $\mathfrak{X}$ and its children $\mathfrak{Ch}_{\mathfrak{X}}$ denoted $(Z^0_{i,\mathfrak{X}})_i$. The CPD that rule X^0 is then expressed in Equation A.2.

$$P(X^0 = +x \mid T^1_{\mathfrak{X}}, (Z^0_{i,\mathfrak{X}})_i) = \left(1 + \frac{P(\neg x)P(T^1_{\mathfrak{X}} \mid \neg x)\prod_i P\left(Z^0_{i,\mathfrak{X}} \mid \neg x\right)}{P(+x)P(T^1_{\mathfrak{X}} \mid +x)\prod_i P\left(Z^0_{i,\mathfrak{X}} \mid +x\right)}\right)^{-1} \tag{A.2}$$

$T^1_{\mathfrak{X}}$	$P(X^0 = +x \mid T^1_{\mathfrak{X}})$
$\neg t$	$\left(1 + \frac{(1-p_{\mathfrak{X}})(1-l_{\mathfrak{X}})}{p_{\mathfrak{X}} f_{\mathfrak{X}}}\right)^{-1}$
$+t$	$\left(1 + \frac{(1-p_{\mathfrak{X}}) l_{\mathfrak{X}}}{p_{\mathfrak{X}}(1-f_{\mathfrak{X}})}\right)^{-1}$

Table A.1: CPD of the variable X^0 when $\mathfrak{X}$ is not involved in any prerequisite relationship. $p_{\mathfrak{X}}$ is the prior probability of $\mathfrak{X}$, $l_{\mathfrak{X}}$ is the learn probability of $\mathfrak{X}$ and $f_{\mathfrak{X}}$ is the forget probability of $\mathfrak{X}$.

The resulting CPD is represented in Table A.2.

$T^1_{\mathfrak{X}}$	$P\left(X^0 = +x \mid T^1_{\mathfrak{X}}, (Z^0_{i,\mathfrak{X}})_i\right)$
$\neg t$	$\left(1 + \frac{(1-p_{\mathfrak{X}}) \times (1-l_{\mathfrak{X}}) \times \left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^+} s_{i,\mathfrak{X}}\right)\left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^\neg} 1 - s_{i,\mathfrak{X}}\right)}{p_{\mathfrak{X}} \times f_{\mathfrak{X}} \times \left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^+} 1 - q_{i,\mathfrak{X}}\right)\left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^\neg} q_{i,\mathfrak{X}}\right)}\right)^{-1}$
$+t$	$\left(1 + \frac{(1-p_{\mathfrak{X}}) \times l_{\mathfrak{X}} \times \left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^+} s_{i,\mathfrak{X}}\right)\left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^\neg} 1 - s_{i,\mathfrak{X}}\right)}{p_{\mathfrak{X}} \times (1-f_{\mathfrak{X}}) \times \left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^+} 1 - q_{i,\mathfrak{X}}\right)\left(\prod_{Z_{i,\mathfrak{X}} \in \mathbf{Z}^\neg} q_{i,\mathfrak{X}}\right)}\right)^{-1}$

Table A.2: CPD of the variable X^0 when $\mathfrak{X}$ is a root in the prerequisite structure. $\mathbf{Z}^+$ is the set of Z auxiliary variables such that $(Z^0_{i,\mathfrak{X}})_i = +z_i$, $\mathbf{Z}^\neg$ is the set of Z auxiliary variables such that $(Z^0_{i,\mathfrak{X}})_i = \neg z_i$. $p_{\mathfrak{X}}$ is the prior probability of $\mathfrak{X}$, $l_{\mathfrak{X}}$ is the learn probability of $\mathfrak{X}$ and $f_{\mathfrak{X}}$ is the forget probability of $\mathfrak{X}$. $q_{i,\mathfrak{X}}$ and $s_{i,\mathfrak{X}}$ are the parameters associated to the influence of the mastery of $\mathfrak{X}$ on $\mathfrak{Ch}_{\mathfrak{X},\mathfrak{i}}$.

When $\mathfrak{X}$ has prerequisite KCs, then X^0 is an AND function of the auxiliary variables between the mastery of $\mathfrak{X}$'s parents $\mathfrak{Pa}_{\mathfrak{X}}$ and $\mathfrak{X}$ denoted $Z^0_{\mathfrak{X},i}$. The resulting CPD is represented in Table A.3.

$\forall i,\ Z^0_{\mathfrak{X},i} = +z_i$	$P\left(X^0 = +x \mid T^1_{\mathfrak{X}}, (Z^0_{\mathfrak{X},i})_i\right)$
0	0
1	1

Table A.3: CPD of the variable X^0 when $\mathfrak{X}$ has prerequisite KCs.

This particular case is one of the specificities making the Gibbs sampling in E-PRISM even easier than in classic tabular CPD Bayesian networks.

A.1.2 Prerequisite nodes in $\mathcal{B}_0$

The auxiliary variables from prerequisite relationships in the initial Bayesian network always have the same Markov blanket. The Markov blanket of $Z^0_{\mathfrak{X},i}$ is composed of the variables $Pa^0_{\mathfrak{X},i}$, X^0, and for $j \neq i$, $Z^0_{\mathfrak{X},j}$. The CPD of the variable $Z^0_{\mathfrak{X},i}$ is expressed in Equation A.3.

$$P(Z^0_{\mathfrak{X},i} = +z_i \mid Pa^0_{\mathfrak{X},i}, X^0, (Z^0_{\mathfrak{X},j})_{j\neq i}) = \left(1 + \frac{P(\neg z_i \mid Pa^0_{\mathfrak{X},i})P(X^0 | \neg z_i, (Z^0_{\mathfrak{X},j})_{j\neq i})}{P(+z_i \mid Pa^0_{\mathfrak{X},i})P(X^0 \mid +z_i, (Z^0_{\mathfrak{X},j})_{j\neq i})}\right)^{-1} \tag{A.3}$$

This expression can be simplified depending on the state of variables in the condition.

If $X^0 = \neg x$, then $P(X^0 \mid \neg z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = 1$ because X^0 is an AND function of the variables $(Z^0_{\mathfrak{X},k})_k$. If in addition $\forall j \neq i, Z^0_{\mathfrak{X},j} = +z_j$, then $P(X^0 \mid +z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = P(X^0 \mid \neg z_i, (Z^0_{\mathfrak{X},j})_{j\neq i})$, otherwise $P(X^0 \mid +z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = 1$.

If $X^0 = +x$, then $P(X^0 \mid \neg z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = 0$. If in addition $\exists j \neq i, Z^0_{\mathfrak{X},j} = \neg z_j$, then $P(X^0 \mid +z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = P(X^0 \mid \neg z_i, (Z^0_{\mathfrak{X},j})_{j\neq i})$, otherwise $P(X^0 \mid +z_i, (Z^0_{\mathfrak{X},j})_{j\neq i}) = 1$.

From the detailed cases above, one can resume the CPD in Table A.4.

X^0	$\forall j \neq i, Z^0_{\mathfrak{X},j} = +z_j$	$Pa^0_{\mathfrak{X},i}$	$P(Z^0_{\mathfrak{X},i} = +z_i \mid Pa^0_{\mathfrak{X},i}, X^0, (Z^0_{\mathfrak{X},j})_{j\neq i})$
0	0	0	$s_{\mathfrak{X},i}$
0	0	1	$1 - q_{\mathfrak{X},i}$
0	1	0	0
0	1	1	0
1	0	0	$s_{\mathfrak{X},i}$
1	0	1	$1 - q_{\mathfrak{X},i}$
1	1	0	1
1	1	1	1

Table A.4: CPD of the variable $Z^0_{\mathfrak{X},i}$.

A.2 Transition Bayesian network

We now take a look at the variables of the transition Bayesian network $\mathcal{B}_\rightarrow$. $\mathcal{B}_\rightarrow$ is composed of both mastery nodes X^t, the prerequisite nodes $(Z^t_{\mathfrak{X},i})_i$ and the transition node $T^t_{\mathfrak{X}}$ for each

KC $\mathfrak{X}$ and for $t > 0$.

A.2.1 Mastery nodes in $\mathcal{B}_{\rightarrow}$

In the manner of the mastery nodes in the initial Bayesian network, the mathematical expression of the CPD for the mastery nodes in the transition Bayesian network can be simplified depending on the position of the KC in the prerequisite structure of the domain model.

$\mathfrak{X}$ with no prerequisite KC

When $\mathfrak{X}$ has no prerequisite KC, the Markov blanket of X^t is only composed of the variables $T^t_{\mathfrak{X}}$, $T^{t+1}_{\mathfrak{X}}$, and if $\mathcal{X}$ is the prerequisite KC of other KCs $\forall j, Z^t_{j,\mathfrak{X}}$. The CPD that rules X^t is expressed in Equation A.4.

$$P(X^t = +x \mid T^t_{\mathfrak{X}}, T^{t+1}_{\mathfrak{X}}, (Z^t_{j,\mathfrak{X}})_j) = \left(1 + \frac{P(\neg x \mid T^t_{\mathfrak{X}})P(T^{t+1}_{\mathfrak{X}} \mid \neg x)P((Z^t_{j,\mathfrak{X}})_j \mid \neg x)}{P(+x \mid T^t_{\mathfrak{X}})P(T^{t+1}_{\mathfrak{X}} \mid +x)P((Z^t_{j,\mathfrak{X}})_j \mid +x)}\right)^{-1} \tag{A.4}$$

X^t is a AND function of the variable $T^t_{\mathfrak{X}}$, so $P(+x|T^t_{\mathfrak{X}} = \neg t) = 0$ and $P(+x|T^t_{\mathfrak{X}} = +t) = 1$. The resulting CPD is represented in Table A.5.

$T^t_{\mathfrak{X}}$	$P(X^t = +x \mid T^t_{\mathfrak{X}}, T^{t+1}_{\mathfrak{X}}(Z^t_{j,\mathfrak{X}})_j)$
$\neg t$	0
$+t$	1

Table A.5: CPD of the variable X^t for $t > 0$ when $\mathfrak{X}$ has no prerequisite KC.

$\mathfrak{X}$ with prerequisite KCs

On the contrary, when $\mathfrak{X}$ has prerequisite KCs, the Markov blanket also contains the variables $(Z^t_{\mathfrak{X},i})_i$. The CPD that then rules X^t is expressed in Equation A.5.

$$P(X^t = +x \mid T^t_{\mathfrak{X}}, T^{t+1}_{\mathfrak{X}}, (Z^t_{\mathfrak{X},i})_i, (Z^t_{j,\mathfrak{X}})_j) = \left(1 + \frac{P(\neg x \mid T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},i})_i)P(T^{t+1}_{\mathfrak{X}} \mid \neg x)P((Z^t_{j,\mathfrak{X}})_j \mid \neg x)}{P(+x \mid T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},i})_i)P(T^{t+1}_{\mathfrak{X}} \mid +x)P((Z^t_{j,\mathfrak{X}})_j \mid +x)}\right)^{-1} \tag{A.5}$$

If $T^t_{\mathfrak{X}} = \neg t$ or if there exists $\mathfrak{Pa}_i$ such that $Z^t_{\mathfrak{X},i} = \neg z_i$, then $P(+x|T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},i})_i) = 0$ because of the deterministic AND function. Otherwise, $P(+x|T^t_{\mathfrak{X}}, (Z^t_k)_k) = 1$. The resulting CPD is represented in Table A.6.

$T^t_{\mathfrak{X}}$	$\forall i, Z^0_{\mathfrak{X},i} = +z_i$	$P(X^t = +x \mid T^t_{\mathfrak{X}}, T^{t+1}_{\mathfrak{X}}, (Z^t_{\mathfrak{X},i})_i, (Z^t_{j,\mathfrak{X}})_j)$
$\neg t$	0	0
$\neg t$	1	0
$+t$	0	0
$+t$	1	1

Table A.6: CPD of the variable X^t for $t > 0$ when $\mathfrak{X}$ has prerequisite KCs.

A.2.2 Transition nodes in $\mathcal{B}_{\rightarrow}$

$\mathfrak{X}$ with no prerequisite KC

When $\mathfrak{X}$ has no prerequisite KCs, then the transition variable $T^t_{\mathfrak{X}}$ has only the variables X^{t-1} and X^t in its Markov blanket. The CPD that then rules X^t is expressed in Equation A.6.

$$P(T^t_{\mathfrak{X}} = +t \mid X^{t-1}, X^t) = \left(1 + \frac{P(\neg t \mid X^{t-1})P(X^t \mid \neg t)}{P(+t \mid X^{t-1})P(X^t \mid +t)}\right)^{-1} \tag{A.6}$$

If $X^t = 0$, then $P(X^t|+t) = 0$ and $P(X^t|\neg t) = 1$. Otherwise, if $X^t = 1$, then $P(X^t|+t) = 1$ and $P(X^t|\neg t) = 0$. The resulting CPD is represented in Table A.7.

X^t	$P(T^t_{\mathfrak{X}} = +t \mid X^{t-1}, X^t)$
0	0
1	1

Table A.7: CPD of the variable $T^t_{\mathfrak{X}}$ for $t > 0$ when $\mathfrak{X}$ has no prerequisite KC.

$\mathfrak{X}$ with prerequisite KCs

When $\mathfrak{X}$ has prerequisite KCs, then the Markov blanket of $T^t_{\mathfrak{X}}$ also contains the variables $\forall i,\ Z^t_{\mathfrak{X},i}$. The CPD that rules X^t is expressed in Equation A.7.

$$P(T^t_{\mathfrak{X}} = +t \mid X^{t-1}, X^t, (Z^t_{\mathfrak{X},i})_i) = \left(1 + \frac{P(\neg t \mid X^{t-1})P(X^t \mid \neg t, (Z^t_{\mathfrak{X},i})_i))}{P(+t \mid X^{t-1})P(X^t \mid +t, (Z^t_{\mathfrak{X},i})_i)}\right)^{-1} \tag{A.7}$$

If $X^t = \neg x$, $P(X^t|\neg t, (Z^t_{\mathfrak{X},i})_i) = 1$. If in addition we have $\forall i,\ Z^t_{\mathfrak{X},i} = +z_i$, then $P(X^t|t, (Z^t_{\mathfrak{X},i})_i) = 0$. If $X^t = 0$ and $\exists i$ such that $Z^t_{\mathfrak{X},i} = \neg z_i$, then $P(X^t|t, (Z^t_{\mathfrak{X},i})_i) = 0$.

If $X^t = +x$ and if $\forall i,\ Z^t_{\mathfrak{X},i} = +z_i$, then $P(X^t|\neg t, (Z^t_{\mathfrak{X},i})_i) = 0$ and the numerator is strictly positive. However, if $X^t = +x$ and $\exists i$ such that $Z^t_{\mathfrak{X},i} = 0$, then $P(X^t|t, (Z^t_{\mathfrak{X},i})_i) = P(X^t|\neg t, (Z^t_{\mathfrak{X},i})_i)$. The resulting CPD is represented in Table A.8.

X^t	$\forall i,\ Z^t_{\mathfrak{X},i} = +z_i$	X^{t-1}	$P(T^t_{\mathfrak{X}} = +t \mid X^{t-1}, X^t, (Z^t_{\mathfrak{X},i})_i)$
0	0	0	$l_{\mathfrak{X}}$
0	0	1	$1 - f_{\mathfrak{X}}$
0	1	0	0
0	1	1	0
1	0	0	$l_{\mathfrak{X}}$
1	0	1	$1 - f_{\mathfrak{X}}$
1	1	0	1
1	1	1	1

Table A.8: CPD of the variable $T^t_{\mathfrak{X}}$ for $t > 0$ when $\mathfrak{X}$ has prerequisite KCs. $l_{\mathfrak{X}}$ and $f_{\mathfrak{X}}$ are the parameters associated to the causal influence of X^{t-1} on $T^t_{\mathfrak{X}}$.

A.2.3 Prerequisite nodes in $\mathcal{B}_{\rightarrow}$

Let's now consider the prerequisite auxiliary variables in the transition Bayesian network $\mathcal{B}_{\rightarrow}$. The auxiliary variable $Z^t_{\mathfrak{X},i}$ between $Pa^t_{\mathfrak{X},i}$ and X^t all have the following variables in their Markov blanket: $Pa^t_{\mathfrak{X},i}$, X^t, $T^t_{\mathfrak{X}}$ and $\forall j \neq i,\ Z^t_{\mathfrak{X},j}$. The CPD of $Z^t_{\mathfrak{X},i}$ is expressed in Equation A.8.

$$P(Z^t_{\mathfrak{X},i} = +z_i \mid Pa^t_{\mathfrak{X},i}, X^t, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = \left(1 + \frac{P(\neg z_i \mid Pa^t_{\mathfrak{X},i})P(X^t \mid \neg z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i})}{P(+z_i \mid Pa^t_{\mathfrak{X},i})P(X^t \mid +z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i})}\right)^{-1} \tag{A.8}$$

As we did for the auxiliary variables in $\mathcal{B}_0$, we simplify Equation A.8 for particular states of the variables in the condition.

If $X^t = \neg x$, then $P(X^t \mid \neg z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = 1$. If in addition $T^t_{\mathfrak{X}} = \neg t$ or $\exists j \neq i, Z^t_{\mathfrak{X},j} = \neg z_j$, then $P(X^t \mid +z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = 1$, otherwise $P(X^t \mid +z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = P(X^t \mid \neg z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i})$

If $X^t = +x$, then $P(X^t \mid \neg z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = 0$. If in addition $T^t_{\mathfrak{X}} = \neg t$ or $\exists j \neq i, Z^t_{\mathfrak{X},j} = \neg z_j$, then $P(X^t \mid +z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = P(X^t \mid \neg z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i})$, otherwise $P(X^t \mid +z_i, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i}) = 1$.

From all of the detailed cases above, one can resume the CPD in Table A.9.

X^t	$\left(\forall j \neq i, Z^t_{\mathfrak{X},j} = +z_j\right) \cap \left(T^t_{\mathfrak{X}} = +t\right)$	$Pa^t_{\mathfrak{X},i}$	$P(Z^t_{\mathfrak{X},i} = +z_i \mid Pa^t_{\mathfrak{X},i}, X^t, T^t_{\mathfrak{X}}, (Z^t_{\mathfrak{X},j})_{j\neq i})$
0	0	0	$s_{\mathfrak{X},i}$
0	0	1	$1 - q_{\mathfrak{X},i}$
0	1	0	0
0	1	1	0
1	0	0	$s_{\mathfrak{X},i}$
1	0	1	$1 - q_{\mathfrak{X},i}$
1	1	0	1
1	1	1	1

Table A.9: CPD of the variable $Z^t_{\mathfrak{X},i}$.

Appendix B

Characteristics of the real-world sub-datasets

B.1 ASSISTments12

$\mathfrak{A}$ $\mathfrak{B}$	MDI ASI	MF ASF	ASI ASF	ASPD MDPD	DF ASF	E ASI
Transactions	126 585	235 334	198 950	298 036	213 120	209 697
Learners	9 131	11 564	14 379	14 538	10 784	11 277
Items	2276	2280	2060	5054	4617	4638
% correctness	0.76	0.77	0.73	0.70	0.68	0.68
Items per KC	1138	1140	1030	2527	2308.5	2319
Transactions per learner	10	12	7	10	12	11

Table B.1: Characteristics of the sub-datasets extracted from *ASSISTments12*.

B.2 ASSISTments17

$\mathfrak{A}$ $\mathfrak{B}$	P ES	PF P	EFDP P	PF ES	A P	PT P
Transactions	71 269	78 624	62 851	40 881	63 258	62 982
Learners	1 561	1 554	1 553	1 456	1 555	1 608
Items	187	256	166	161	164	148
% correctness	0.36	0.39	0.39	0.37	0.36	0.36
Items per KC	93.5	128	83	80.5	82	74
Transactions per learner	32	35.5	28	19	27	27

Table B.2: Characteristics of the sub-datasets extracted from *ASSISTments17*.

B.3 Eedi

$\mathfrak{A}$ $\mathfrak{B}$	MLCM FHCF	PNPF FHCF	PNPF MLCM	MMD VNP	MAS VNP	MMD MAS
Transactions	4031	4269	4110	9308	10 288	8696
Learners	818	795	824	1005	1030	822
Items	476	428	462	1416	1513	1489
% correctness	0.68	0.71	0.67	0.79	0.79	0.79
Items per KC	238	214	231	708	756.5	744.5
Transactions per learner	4	4	4	4	5	5

Table B.3: Characteristics of the sub-datasets extracted from *Eedi2020*.

B.4 Kartable

$\mathfrak{A}$ $\mathfrak{B}$	CF Solve	Root OR	Solve Root	CF Chart	Solve Chart	OR D
Transactions	20 487	18 000	18 556	18 356	13 419	11 631
Learners	3190	2462	3139	3233	2284	1423
Items	13	18	20	12	13	10
% correctness	0.63	0.77	0.75	0.64	0.72	0.80
Items per KC	6.5	9	10	6	6.5	5
Transactions per learner	5	5	5	4	4	6

Table B.4: Characteristics of the sub-datasets extracted from *Kartable*.

Appendix C

RMSE values computed on real-world sub-datasets

C.1 RMSE values obtained from *ASSISTments12* sub-datasets

$\mathfrak{A}$	MDI	MF	ASI	ASPD	DF	E
$\mathfrak{B}$	ASI	ASF	ASF	MDPD	ASF	ASI
$e\Delta_{\varnothing}$	0.345	0.425	0.417	**0.372**	0.446	**0.389**
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.337	0.426	**0.392**	0.379	0.439	0.396
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	**0.327**	**0.420**	0.419	0.374	**0.434**	0.401

Table C.1: RMSE values obtained from the training set of sub-datasets of *ASSISTments12*.

C.2 RMSE values obtained from *Eedi2020* sub-datasets

$\mathfrak{A}$	MLCM	PNPF	PNPF	MMD	MAS	MMD
$\mathfrak{B}$	FHCF	FHCF	MLCM	VNP	VNP	MAS
$e\Delta_{\varnothing}$	**0.449**	0.455	**0.460**	0.409	**0.396**	0.414
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	**0.449**	**0.444**	0.461	0.402	0.409	**0.402**
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.450	0.450	0.470	**0.399**	0.403	0.408

Table C.2: RMSE values obtained from the training set of sub-datasets of *Eedi2020*.

C.3 RMSE values obtained from *Kartable* sub-datasets

$\mathfrak{A}$ $\mathfrak{B}$	CF Solve	Root OR	Solve Root	CF Chart	Solve Chart	OR D
$e\Delta_{\varnothing}$	**0.421**	0.372	**0.359**	**0.389**	0.389	**0.286**
$e\Delta_{\mathfrak{A}\to\mathfrak{B}}$	0.424	**0.359**	0.366	0.407	0.376	0.296
$e\Delta_{\mathfrak{B}\to\mathfrak{A}}$	0.422	0.364	0.364	0.398	**0.368**	0.294

Table C.3: RMSE values obtained from the training set of sub-datasets of *Kartable*.