KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Stochastic Relational Processes and Models

Learning and Reasoning

Ingo Thon

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

October 2011

Stochastic Relational Processes and Models

Learning and Reasoning

Ingo Thon

Jury:
Prof. Dr. ir. Willy Sansen (chair)
Prof. Dr. Luc De Raedt (promotor)
Prof. Dr. ir. Hendrik Blockeel
Prof. Dr. ir. Philip Dutré
Prof. Dr. Jesse Davis
Dr. James Cussens
(University of York)
Dr. Kristian Kersting
(University of Bonn/Fraunhofer IAIS)

Dissertation presented
in partial fulfillment
of the requirements for
the degree of Doctor
in Engineering

October 2011

D/2011/7515/130
978-94-6018-429-1

Abstract

In order to solve real-world tasks, intelligent machines need to be able to act in noisy worlds where the number of objects and the number of relations among the objects varies from domain to domain. Algorithms that address this setting fall into the subfield of artificial intelligence known as *statistical relational artificial intelligence* (StaR-AI).

While early *artificial intelligence* systems allowed for expressive relational representations and logical reasoning, they were unable to deal with uncertainty. On the other hand, traditional probabilistic reasoning and machine learning systems can capture the inherent uncertainty in the world, but employ a purely propositional representation and are unable to capture the rich, structured nature of many real-world domains.

StaR-AI encompasses many strains of research within artificial intelligence. One such direction is statistical relational learning which wants to unify relational and statistical learning techniques. However, only a few of these techniques support decision making processes.

This thesis advances the state-of-the-art in statistical relational learning by making three important contributions. The first contribution is the introduction of a novel representation, called causal probabilistic time-logic (CPT-L) for stochastic relational processes. These are stochastic processes defined over relational state-spaces and they occupy an intermediate position in the expressiveness/efficiency trade-off. By focusing on the sequential aspect and deliberately avoiding the

complications that arise when dealing with hidden states, the algorithms for inference and learning for CPT-L are more efficient than those of general purpose statistical relational learning approaches. The second contribution is that we show how to adapt and generalize the algorithms developed for CPT-L so that they can be used to perform parameter estimation in the probabilistic logic programming language ProbLog. The final contribution of this thesis is a decision theoretic extension of the ProbLog language that allows to represent and to solve decision problems.

Acknowledgments

To pursue a Ph.D. was the biggest challenge in my life so far. I would like to use these pages to express my gratitude to all the people that were involved in this project and helped me achieving this goal

First of all, I would like to express my gratitude to my supervisor, Luc De Raedt who was abundantly helpful and offered invaluable assistance, support and guidance throughout the last years. He always challenged me with interesting research questions, starting from my "Studienarbeit", continuing with my diploma thesis both back in Freiburg and throughout my entire Ph.D. studies in Belgium – I have learned a lot from him. When he asked me what I would like to do after my diploma, I answered that I would like to live some time abroad... He made this possible, and I did not only get acquainted with a new culture and language but also made a lot of new friends.

I would like to thank the members of my jury Hendrik Blockeel, Philip Dutré Jesse Davis James Cussens Kristian Kersting for their time and helpful suggestions on the manuscript, which greatly helped improving the text. Furthermore, I would like to thank Willy Sansen for chairing the jury.

Jesse Davis, Bernd Gutmann, Niels Landwehr, Wannes Meert, Guy Van den Broeck, Albrecht Zimmermann read parts of the manuscript and I especially would like to thank them for their time, efforts and their feedback.

I would like to thank Kristian Kersting, who was not only member of my jury but also influenced me a lot. His enthusiasm was one of the reasons for me to start working statistical relational learning (SRL). Even though, he changed his plans and did not come to Leuven he always provided me with advice and thorough feedback on my research. Most importantly, he always forced me to not forget about the rest of SRL. Furthermore, his immense knowledge about all the different research was always a touchstone for my ideas, and a great source of inspiration – although his vast knowledge can sometime be challenging.

Two colleagues I would like to thank specifically as they influenced my path in research a lot are Niels Landwehr and Bernd Gutmann. Niels's very systematic and highly structured approach to writing papers, running experiments, and programming influenced me beyond our joint papers and I greatly enjoyed working together with him. Bernd's knowledge of science in general and especially math but also his eye for details was always inspiring and helpful. Furthermore I would like to thank people I co-authored papers with during my Ph.D. studies and that have not been mentioned above: Laura-Andrea Antanas, Maurice Bruynooghe, Angelika Kimmig, Guy Van den Broeck and Martijn van Otterlo.

Of course there are many more people I would like to thank: Kurt Driessens, Robby Goetschalckx and Guy Van den Broeck for being such great office mates and continuously trying to improve my Flemish by "praten" to each other, but also all my other former office mates. I enjoyed many interesting conversation with the almost inseparable Albrecht Zimmermann and Björn Bringmann, as well as with Fabrizio Costa, who often approached me with "Here is the idea....". I am deeply indebted to Jesse Davis for him providing me invaluable support during the final stage of my Ph.D. studies.

There are many more people to thank but not much more space left, I would like to express my gratitude to the whole machine learning group in Leuven and the people in the background, such as the secretary and the system administrators. You made working here very enjoyable.

I gratefully acknowledge the financial support received for the work performed during my thesis from the K.U. Leuven, the FWO project: Relational action and activity learning, the GOA/08/008, Probabilistic Logic Learning and by the European Community's Seventh Framework Programme under grant agreement First-MM-248258.

I would also like to thank my friends from the Black Forest Toastmasters of Brussels. I did not only learn a lot about public speaking, goal setting and leadership, but you also about personal development.

A special thanks also to my parents and brothers. You provided me with a home that allowed me to grow up in a protected environment and prepared me for the serious side of life (and science).

Most importantly, I would like to thank my girlfriend Ursula Rings. Ohne Deine Unterstützung wäre ich in den letzten Jahren nur allzu oft an mir selbst oder an meiner Arbeit verzweifelt. Danke für Deine Liebe, Deinem konstanten Zuspruch und dafür, dass Du mich erträgst.

Ingo Thon
Leuven, October 2011

Contents

List of Figures

LF

List of Algorithms

LA

List of Symbols

LS

$\mathtt{X}, \mathtt{Y}, \ldots$	variable (uppercase)
$\mathtt{c}$	constant (lowercase)
$\mathtt{p}/n$	predicate p with arity n
$\mathcal{F}$	set of probabilistic facts
$\mathcal{BK}$	background knowledge
$p :: f$	probabilistic fact f labeled with probability p
$? :: f$	decision fact indicated by label ?
$\mathcal{F}_l$	set of labeled facts
ω	possible world
Ω	set of possible worlds
$\mathcal{F}_\omega$	set of facts true in ω
$\mathcal{T}$	theory $\mathcal{F} \cup \mathcal{BK}$
$[\![q]\!]$	worlds $\mathcal{F}_\omega$, which entail q, this is $\mathcal{F}_\omega \cup \mathcal{BK} \models q$
$(f^{(i)})_{i \in \mathbb{N}}$	series $f^{(1)}, f^{(2)}, \ldots$
$(P^{(i)}_{\mathcal{F}})_{i \in \mathbb{N}}$	series of distributions over a subset of facts
$P_{\mathcal{F}}$	distribution over all facts
$P_{\mathcal{T}}$	distribution over interpretations
$LH(P)$	least Herbrand model of program P
$X, Y, \ldots$	random variable
$x, y, \ldots$	outcome of random experiment
$\mathbf{X}$	set of random variables (bold font)

$\mathbb{E}[X]$	expectation of X
$\mathbb{E}[X\|Y=y]$	conditional expectation of X given $Y=y$
$\mathrm{EU}(a)$	expected utility of decision a
I_i	interpretation at time point i
$I_{[0:t]}$	sequence of interpretations $I_0, \ldots, I_t$
$\sigma(\cdot)$	selection

1 Overture

INTELLIGENCE is considered as the

> [...] mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. [...] it reflects a broader and deeper capability for comprehending our surroundings – "catching on", "making sense" of things, or "figuring out" what to do.
>
> [Gottfredson, 1997]

Artificial intelligence is the "science and engineering of making intelligent machines" [McCarthy and Hayes, 1987]. The skills required by an intelligent machine can be divided into two broad categories.

The first category of skills can be summarized as the ability to **learn**, which is studied in machine learning, a discipline that develops algorithms that improve their performance with experience.The acquired knowledge is often called the model, allows. It allows one to reason about the probability of past events and likeliness of future events.

The second category of skills can be summarized as the ability to **decide** what to do, and is concerned with deciding which actions to take. An intelligent machine is supposed to act rational [Russell and Norvig, 2003]. Essentially, the machine should make the decisions that maximize the expected utility, that is, the rewards

it expects to obtain. Decision making techniques can employ a model, possibly generated using machine learning techniques.

Finally, the machine must be able to apply all these skills across a multitude of complex environments. Therefore, it needs to be able to represent knowledge at an abstract level and to reason about sets of objects rather than be restricted to reason about specific entities only.

The following example is used to illustrate some of the key concepts of this thesis.

Example 1.1 (Online book seller) *Consider an online book seller. The problem that the store faces is that, in contrast to normal book stores, people do not enter the store and browse through the bookshelves. Instead the store needs to recommend books that might be of interest to customers. In fact, one of the largest online book sellers spent more then 1 billion dollar (*3% *of net sales) on marketing in 2010. In comparison, fulfillment (billing, payment processing) was 2.898 million dollar [Bezos, 2010]. In such a setting, targeted marketing requires the human or system to decide which products to recommend to which customer at any point in time.*

This task, as well as many other tasks, require intelligence and cannot be performed by humans due to the sheer volume of information that needs to be processed in order to learn and the number of decisions that need to be made to solve the task at hand. Similar challenges occur in social network analysis, analysis of purchase profiles, and internet search queries among others. Statistical relational AI (StaR-AI) aims at developing machines that are able to act in noisy, complex environments that consist of multiple objects and relations among them [Kersting et al., 2010]. Thus it aims at developing intelligent machines that can solve these tasks [Poole, 2011]. More formally, it

StaR-AI faces at least there are four main challenges that arise real-world applications.

(chal1) The real world consists of a large number of objects. This includes for example, the number of customers or books the shop is selling.

(chal2) The world is inherently relational, that is, the objects in the world are related to each other. For instance, different customers can be friends, or live in the same household; books can be part of a series, might be written by the same author, or cover the same topic.

(chal3) The real-world is inherently uncertain. Because one customer has bought a pair of books together, every other customer that is interested in one of these books is not necessarily also interested in the other one.

(chal4) Many of these applications are sequential by nature, which may be by the temporal ordering. Books belonging to one series are also often bought in the corresponding orders, it is possible to exploit such pattern, if the sequential nature of a problem is taken into account explicitly. The order of events is not just another relation as it implies that there is a large amount of information available in the form of the history. Furthermore, explicitly representing and dealing with time may result in improved performance.

Finding solutions to these challenges is not only of scientific interest but also also economically interesting, as it can reduce the marketing costs of companies as well as increase the return on investment. While the tools developed throughout this thesis do not provide operational solutions for an online bookseller, the applications used in the different chapters are instances of sub-problems a book seller might face: we predict what people do next based on their social network (Chapter 4), we predict whether a person is a professor or a student based on the available information (Chapter 5), and finally we study how to decide upon whom to market to and whom not to market to (Chapter 6).

We will now illustrate several aspects of our techniques using the online bookseller example. We also give more details on how the solution needs to combine learning, logic and probability theory, and relate it to existing research in the field of statistical relational artificial intelligence.

1.2 Context

This thesis fits into statistical relational AI (StaR-AI). Star-AI lies at the intersection of probability theory [Wasserman, 2003], logical reasoning [Flach, 1994], relational modeling [Ramakrishnan and Gehrke, 2003], machine learning [Mitchell, 1997], and decision making under uncertainty [Feldman and Sproull, 1977; Bacchus et al., 1999].

The term StaR-AI serves as an umbrella term that encompasses different research efforts. Most of the existing works within the envisioned field of StaR-AI focus either on decision making or on learning. Decision making has largely been studied in the context of relational Markov decision processes [Boutilier et al., 2001] and relational reinforcement learning [Dzeroski et al., 2001]. The field concerned with learning is probabilistic logic learning [De Raedt and Kersting, 2003; De Raedt, 2008; Kersting, 2006], also called statistical relational learning (SRL) [Getoor and Taskar, 2007].

The approach to statistical relation learning, on which we focus in this thesis, is probabilistic programming [Koller et al., 1997; Sato and Kameya, 1997; Poole,

1997], which we will use to address reasoning and learning problems that arise in StaR-AI.

Probabilistic programming languages combine general purpose programming languages with statistical modelling techniques. The advantage of using a programming language is that this ensures that the specification of the programs is precise and, therefore, that the semantics is well understood. The family of probabilistic programming languages considered in this thesis is based on logic programming. These languages allow easy communication of the semantics of a model to non-programmers as they make use of logical statements with a declarative semantics which are easy for humans to interpret.

In the following we explain how and why the four challenges occur in real-world problems and will also explain how probabilistic logic learning takes on these challenges.

1.2.1 Learning

Developing models representing a domain by hand is often a tedious task. Hence it is desirable that machines automatically learn models from experience. Especially, when there is a large number of objects (cf. chal1) the expert needs to model a large amount of information and existing knowledge.

The process of learning as used in this thesis can be regarded as condensing the available experience about a large number of objects in a single abstract model, thereby reducing the complexity of the information.

Example 1.2 *In the example of the bookseller the objects are, for example, the books and the customers. Building a model stating that a customer buys a book X because of a reason Y is infeasible for humans due to the sheer number of books. By using machine learning such a model can be automatically generated. One simple approach would be to count the number of times books have been bought together and to always recommend the book that has been bought together most often with the one the customer is about to buy.*

1.2.2 Logic and Relations

The second aspect that is important to the work presented in this thesis is logic and relations (cf. chal2). Traditional machine learning techniques expect data to come in the form of feature vectors where each training example is described by a set of attributes.

Example 1.3 *A typical purchase record could consist of attributes or features that describe the customer such as age and location as well as features of the books he buys such as price and subject. This form of representation is very limited as it cannot capture relations such as those given by the customer's social network, for example. that the book was bought by one of his friends, or that the author of the book is the supervisor or research colleague of the customer, or that the customer previously bought another volume from the same series.*

Relational logic can represent knowledge about the entities by means of relations in an elegant manner. Additionally, logic represents knowledge in a declarative way.

Example 1.4 *The following first order formula encodes knowledge about purchase patterns compactly*

$$\forall Book, Customers :: buys(Customer, AnotherBook) \wedge$$
$$incollection(AnotherBook, Collection) \wedge incollection(Book, Collection)$$
$$\rightarrow buys(Customer, Book) \ .$$

This formula can be read as follows: for all ($\forall$) Books and Customers, if the Customer buys AnotherBook from the Collection containing Book, this implies ($\rightarrow$) that he will also buy Book.

The statement above is a Horn-Clause. Such clauses form the basis of logic programming. They are declarative, that is, easy to understand to humans.

1.2.3 Probabilities

Logical sentences express certain knowledge, whereas nothing is inherently certain in the real world (cf. chal3). Therefore, we use probabilistic logics which represent uncertainty by associating a probability to each formula that indicates the number of cases in which the formula holds on average.

Example 1.5 *Reconsider the rule that a customer will buy all books from the same collection. This rule holds most, though not all, of the time. If, for example, this regularity holds 90% of the time, we could represent it with a probabilistic logic*

formula as follows

$$\forall Book, Customers :: buys(Customer, AnotherBook) \wedge$$
$$incollection(AnotherBook, Collection) \wedge incollection(Book, Collection)$$
$$\rightarrow 0.9 :: buys(Customer, Book) \ .$$

1.2.4 Sequential ordering

A final aspect is that in many applications the order of events matters (cf. chal4). This is due to the fact that two events are often correlated, as one event may cause the other one. Reconsider the example of the book store, and the rule (Example 1.5) that a person tends to buy multiple volumes from the same collection. This regularity depends very much on the volume bought. For example, analyzing the entire purchase history will probably support this rule. Yet, in most cases the second volume will be bought at the same time or after the first volume whereas it is much less likely that the second volume is purchased before the first volume.[1] This thesis focuses on such temporal sequences, whereas in machine learning different kinds of sequences are considered, for example, DNA which is a sequence of genes.

1.2.5 Related Work

Star-AI and in particular SRL have contributed a wide variety of representations that combine relational representations with probabilities.

Many systems are based on the observation that existing probabilistic models cannot cope with rich relational data. This resulted into the idea of *knowledge-based model construction* (KBMC), which employs expressive relational representations but reduces to well-known propositional systems, for which efficient inference algorithms are available, and extends their representation with relational concepts [Wellman et al., 1992]. The following approaches construct Bayesian networks [Pearl, 1988] relational Bayesian networks [Jaeger, 1997], probabilistic relational models [Friedman et al., 1999], Bayesian logic programs [Kersting and De Raedt, 2007], and CLP($\mathcal{BN}$) [Santos Costa et al., 2008]. Other well-known formalisms that follow this same basic principle include Markov logic networks [Richardson and Domingos, 2006], and relational dependency networks [Neville and Jensen, 2007].

[1] Personal anecdote: Amazon keeps recommending "Diskrete Strukturen 1" to the author of this thesis because he bought "Diskrete Strukturen 2".

Many real-world applications such as part-of-speech tagging require modelling distributions over sequences. All the aforementioned systems allow to model such distributions by explicitly representing the sequence relation, however, the state spaces of dynamical systems are typically very large. This is due to the exponential growth of the number of potential sequences with increasing sequence length. Furthermore, the random variables or attributes of a dynamic system tend to correlate already for very short sequences [Boyen and Koller, 1998], which makes inference often intractable. This problem is not specific to relational representations but also in various specialized propositional sequence models, KBMC methods also exist that employ these specialized propositional models. Examples are Markov models [Markov, 1907], which have been upgraded to relational Markov models (RMM) [Anderson et al., 2002], hidden Markov models [Rabiner, 1989], which have not been upgraded to logical hidden Markov models (LoHMM) [Kersting et al., 2006], and finally conditional random fields [Lafferty et al., 2001], which have been upgraded to TildeCRF [Gutmann and Kersting, 2006].

Another approach to developing probabilistic, relational representations is *probabilistic programming*. Probabilistic programming augments programming languages with probabilistic concepts. The semantics of many probabilistic languages that are based on logic programming are rooted in the distributions semantics [Sato, 1995]. The idea of these languages is to represent the probabilistic part of the model by a set of independent distributions. A logic program combines these distributions to define a joint distribution over possible worlds. Examples for such languages are PRISM [Sato and Kameya, 1997], ICL [Poole, 2008], CP-Logic [Vennekens et al., 2006], and ProbLog [De Raedt et al., 2007]. Additionally, approaches, such as IBAL [Pfeffer, 2001], or Church [Goodman et al., 2008], exist that make use of functional programming languages. A language based on imperative concepts is BLOG [Milch et al., 2005].

1.3 Thesis Contributions and Roadmap

The overall goal of this thesis is to develop algorithms and representations that supports learning and decision making in the context of probabilistic logic programming languages, and are able to cope with the four challenges (chal1-chal4).

This thesis makes three contributions towards these goals: (1) we develop a model that allows one to represent probabilistic models of sequences and provides efficient inference and learning algorithms, (2) we provide a parameter learning algorithm for the language ProbLog which is able to use examples in the form of partial interpretations, and (3) we extend the ProbLog language with decision theoretic concepts and provide the required inference algorithms for this.

The first contribution is concerned with modelling probability distributions over sequences, which is important in many real-world applications. On the one hand, applying the exact inference techniques of general SRL approaches is impossible due to the large number of possible sequences although this situation can sometimes be alleviated by the use of approximate techniques [Kersting et al., 2009]. Approaches within SRL tailored toward sequences, for example, LoHMMs and RMM, on the other hand, allow for exact inference but are limited to a single probabilistic process. However, in most applications, the agents and the environment interact in a stochastic manner. A similar shortcoming has been identified in the decision making or planning community [Sanner, 2010]. Nevertheless, the proposed solution still requires the number of processes to be fixed. For modelling sequential in this thesis restricts the possible models so that that exact inference is still possible. We base our model on a general probabilistic logic-programming language and limit the influence of hidden information to a single time-step. This allows for efficient inference and learning techniques.

The second contribution aims at learning the parameters of probabilistic logic programs. Parameter learning reduces the time that a domain experts must spend, as he only needs to develop the program. Furthermore, the parameters learned from data often yield better results than the ones defined by hand. In previous works [Sato and Kameya, 2001; Cussens, 2001; Muggleton, 2002; Gutmann et al., 2008], learning parameters of probabilistic logic languages the training examples state whether that a particular fact is true or the probability that the fact is true. This often requires one to define the models in a non-generative and therefore less intuitive way. In contrast, our algorithm learns from partial interpretations, these are world descriptions, which is often more intuitive and can also contain more information.

The third contribution integrates decision making concepts within statistical relational representations. This integration has been mainly studied in the context of relational reinforcement learning [Dzeroski et al., 2001] and relational Markov decision processes [Boutilier et al., 2001]. As a result, the representations and algorithms are typically tailored towards sequential domains with a single source of randomness, e.g., a single actor. However, in relational domains like social networks, the decisions need to be factored to achieve compact representations. This issue has been addressed for Markov logic [Nath and Domingos, 2009], but the solver is based on a greedy hill-climbing approach. Factored representation for propositional data has been studied in the context of influence diagrams [Howard and Matheson, 1984/2005]. Furthermore, relational representations with factored decision have been studied for probabilistic programs [Poole, 1997; Pfeffer, 2001] but efficient algorithms for solving decision problems are lacking.

This thesis is organized as follows. It starts by introducing basic concepts and existing works that are required in this thesis.

Chapter 2: We introduce probability theory, as well as statistical learning, statistical decision theory, and probabilistic graphical models. We will also review binary and algebraic decision diagrams and logic programming.

Chapter3: We show how probability theory and logic programming are combined in logic programming, which serves as the basis for the representations developed throughout this thesis. We also show how binary decision diagrams can be utilized for the basic inference in logic programming.

The contributions of this thesis are presented in the following three main chapters.

Chapter 4: The first contribution of this thesis is the formalism of causal probabilistic time logic (CPT-L). It is based on a probabilistic logic programming language and tailored to modeling stochastic relational processes. These processes assign a probability to each sequence of relational states, also called worlds. Stochastic relational processes, of the complexity assumed in this work, have received little attention in the literature. Most of the existing approaches cannot handle the four challenges outlined at the beginning of the introduction as CPT-L as they either only allow a fixed numbers of objects, or do not allow for relations, or are restricted in their ability to represent probabilistic aspects of the world. Efficient inference and learning is possible in CPT-L due to a set of assumptions, which are reasonable. Besides showing that the assumptions allow us to solve interesting real-world tasks, we show how to perform approximate inference in cases where the assumptions are partially violated.
This chapter is based on:

Ingo Thon, Niels Landwehr, Luc De Raedt *A simple model for sequences of relational state descriptions.* In Walter Daelemans, Bart Goethals, and Katharina Morik, *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008)*, volume 5211 of LNCS (Lecture Notes In Computer Science), pages 473–488, September 2008. Springer Berlin/Heidelberg. DOI: 10.1007/978-3-540-87481-2_33

Ingo Thon, Niels, Landwehr, Luc De Raedt *Stochastic relational processes: Efficient inference and applications.* Machine Learning, volume 82, issue 2, pages 239-272, 2011. DOI: 10.1007/s10994-010-5213-8

Ingo Thon, Bernd Gutmann, Martijn van Otterlo, Niels Landwehr, Luc De Raedt. *From non-deterministic to probabilistic planning with the help of statistical relational learning*, *Proceedings of the ICAPS Workshop on Planning and Learning*, pages 23–30, Thessaloniki, 20 September 2009.

Ingo Thon. *Don't fear optimality: Sampling for probabilistic-logic sequence models.* In Luc De Raedt, *Proceedings of the 19th International*

Conference on Inductive Logic Programming, LNCS (Lecture Notes in Computer Science) volume 5989, pages 226-233, Springer, 2010 DOI: 10.1007/978-3-642-13840-9_22

Chapter 5: The second contribution generalizes the learning algorithm for CPT-L to general probabilistic programs that can be used in cases where CPT-L's assumptions are violated. The resulting algorithm for learning ProbLog programs is entirely new in that it studies the learning from interpretations setting. This setting is common for graphical models and in statistical relational learning but has so far not been considered in probabilistic programming. The algorithm splits training examples automatically by exploiting independence, and thereby allows for efficient inference. This splitting operation is motivated by the CPT-L framework, which relies on the assumption that future observations are independent of past observations given current observations. This chapter is based on:

Bernd Gutmann, Ingo Thon, and Luc De Raedt. *Learning the Parameters of Probabilistic Logic Programs from Interpretations.* In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases (ECML PKDD 2011)*, volume 6911 of LNCS (Lecture Notes in Computer Science), pages 581–596. Springer Berlin/Heidelberg, 2011. **Winner of the Best Paper Runner up Award in Machine Learning** (599 submissions). DOI: 10.1007/978-3-642-23780-5_47

Bernd Gutmann, Ingo Thon, and Luc De Raedt. *Learning the parameters of probabilistic logic programs from interpretations.* Technical Report CW 584, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, April 2010.

Chapter 6: The techniques developed in Chapter 5 allow us to learn statistical relational models of a given domain. Chapter 6 shows how such a learned model can be used for decision making under uncertainty. We extend ProbLog to DTProbLog that allows for encoding decision problems. We provide a basic exact inference algorithm, which is able to find the optimal decision. We also present two optimizations of this algorithm as well as some approximate solutions.
This chapter is based on:

Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, Luc De Raedt. *DTProbLog: A decision-theoretic probabilistic Prolog*, In Maria Fox, David Poole, *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence* Atlanta, Georgia, USA, 11-15 July 2010, pages 1217-1222, AAAI Press

Chapter 7: concludes this thesis and gives directions for future work.

The Appendix of this thesis contains the proofs to the main theorems and discuss the relationship between CPT-L and a common planning language.

Some of the work performed throughout my Ph.D. research is not covered in this thesis. This includes: 1) the probabilistic rule learner ProbFOIL [De Raedt and Thon, 2011]. ProbFOIL combines principles of the relational rule learner FOIL with ProbLog. This realizes structure learning for PobLog programs by learning the definition of a single predicate. The second work is concerned with extending probabilistic logic programs with continuous distributions [Gutmann et al., 2011b]. The contributions of this work are two-fold. First, we provide a semantics for a very general framework of so-called hybrid probabilistic logic programs. Second, we introduce an approximate inference algorithm that combines forward sampling with backward inference in order to avoid sampling irrelevant distributions and to reduce the rejection rate, which results in faster convergence.

Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. *The magic of logical inference in probabilistic programming.* Theory and Practice of Logic Programming, 11:663–680, 2011. DOI: 10.1017/S1471068411000238

De Raedt, Luc, Ingo Thon. *Probabilistic rule learning*, In Paolo Frasconi, Francesa Alessandra Lisi,*International Conference on Inductive Logic Programming (ILP)*, volume 6489 of LNCS, (Lecture Notes in Computer Science) pages 47-58, Springer Berlin/Heidelberg, 2011. DOI: 10.1007/s10994-010-5213-8

2 Definitions and Background

This chapter provides the necessary background for this thesis. It is organized as follows. First, we give a brief introduction to statistical learning and decision making under uncertainty. This chapter reviews probabilistic graphical models a representation common in both fields.

Second, we review Algebraic Decision Diagrams and Binary Decision Diagrams. Decision Diagrams are a datastructures that are frequently used throughout this thesis as they represent Boolean functions compactly.

Finally, we briefly review logic programming which forms the basis for the representation of the models used in this thesis.

2.1 Statistical Machine Learning and Decision Theory

In this section we will review the basic concepts underlying statistical machine learning and statistical decision theory. We will start by briefly reviewing

the basis of probability theory. Using the introduced notions we will define the task posed by machine learning and decision making. Finally we will introduce probabilistic graphical models.[1] Probabilistic graphical models are a widely accepted framework used in both fields

2.1.1 Probability Theory

The formalization of probability theory goes back to the definitions by Kolmogorov [1933]. For more recent and detailed references, see [Jensen, 2001], or [Neapolitan, 2003]. Probability theory [Wasserman, 2003] deals with experiments with unknown outcomes.

A **sample space** Ω_Y is a set of all possible outcomes of an experiment. An **event** $\mathbf{y}$ is a non-empty subset of Ω_Y. A σ-algebra Σ_Y over a sample space is a non-empty set of events that contains for each pair of events $\mathbf{y}_1, \mathbf{y}_2 \subset \Sigma_Y$ the union $\mathbf{y}_1 \cup \mathbf{y}_2$, furthermore, the inverse event $\Omega_Y \setminus \mathbf{y}$ of each event $\mathbf{y} \in \Sigma_Y$ needs to be in Σ_Y. A function $P : \Sigma_Y \to \mathbb{R}$ that assigns a real number to each event $\mathbf{y} \in \Sigma_Y$ is called a **probability function** if it satisfies the Kolmogorov axioms:

1. The probability of each event is non-negative $P(\mathbf{y}) \geq 0$ for all $\mathbf{y} \subseteq \Sigma_Y$.
2. All possible outcomes are in the sample space $P(\Omega) = 1$.
3. The sum of the probability of each countable, pairwise disjoint sets of events $\mathbf{y}_1, \mathbf{y}_2, \ldots$ equals the probability of the joint event:

$$P(\mathbf{y}_1 \cup \mathbf{y}_2 \cup \ldots) = \sum_{\mathbf{y}_i} P(\mathbf{y}_i).$$

A **random variable** Y is a (measurable) function $Y : \Sigma_Y \to \mathbb{S}$ that assigns an element of $\mathbb{S}$ to each element of the sample space $\mathbf{y} \in \Sigma_Y$. Typically $\mathbb{S}$ is the set of reel numbers $\mathbb{R}$. A random variable Y induces a distribution $P_{\mathbf{Y}}(\mathbf{s}) = P(Y \in \mathbf{s})$.

Several random variables can be combined, e.g., X, Y, the probability distribution is then called **joint probability distribution** which is written as $P(X, Y)$. If knowledge about the value of one random does not influence the probability of another one then they are said to be **independent** and $P(X, Y) = P(X) \cdot P(Y)$. If the value of one of the random variable X is known to be x then the probability of Y is given by the **conditional probability** $P(Y|X = x)$ defined as $P(Y|X = x) = P(Y, X = x) \cdot P(X = x)^{-1}$ if $P(X = x) > 0$. This notation motivates **conditional independence**, where two random variables are independent given the value of a third random variable is known. Two random variables X, Y are

[1] A more detailed introduction to Bayesian networks but also to many other topics relevant for this thesis can be found in [Darwiche, 2008].

said to be conditionally independent given Z if $P(X,Y|Z) = P(X|Z) \cdot P(Y|Z)$. For a random variable Y with numerical sample space Ω_Y the expected value $\mathbb{E}[Y]$ is the average over the value of the domains each weighted by the probability of the corresponding events which is $\mathbb{E}[Y] = \sum_{y \in \Omega_Y} y \cdot P(Y = y)$. We will also use the notion of conditional expectation $\mathbb{E}[Y|X = x] = \sum_{y \in \Omega_Y} y \cdot P(Y = y|X = x)$.

Statistical machine learning can be roughly seen as estimating a probability distribution based on a set of training examples. The first of the two main tasks we study in this thesis is *learning*[2] *the parameter(s) of a distribution.* A parametric distribution describes a family of distributions where the individual members can be selected by means of a parameter. If the *training examples* $\mathbf{e} = \{e_1, \ldots, e_n\}$, are drawn independently from identical distribution (i.i.d.) and hence $P(\mathbf{e}) = \prod_{e_i \in \mathbf{e}} P(e_i)$, then the task can be formalized as **max-likelihood parameter estimation** where the goal is to find the parameters $\hat{\theta}$ such that:

$$\hat{\theta} = \arg\max_{\theta} P_\theta(\mathbf{e}) = \arg\max_{\theta} \prod_{e_i \in \mathbf{e}} P_\theta(e_i) \ .$$

Example 2.1 *Let us assume the training examples represent how many days* (k) *one needs to wait until it rains. This can be modeled using the geometric distribution, which has the real valued parameter* θ *and* $P_\theta(k) = (1-\theta)^{k-1} \cdot \theta$*. For a set of training examples* $\mathbf{e} = \{5, 2, 9, 5, 2, 7\}$ *the parameter can be estimated as the multiplicative inverse of the sample mean* $\hat{\theta} = \frac{n}{\sum e_i} = 0.2$*.*

The second main task studied in this thesis is decision making under uncertainty. In decision making an agent has to choose which decision to make from a set of possible options $\mathbf{d} = \{d_1, \ d_2, \ \ldots\}$. In the simplest setting the agent selects one of several possible worlds $w_i \in \Omega$. This choice must maximize the utility of the agent is maximized, where the **utility** is defining in terms of a real valued function $u : \Omega \to \mathbb{R}$. An agent is said to act rationally if he picks the world with the highest utility. In **statistical decision making** [Berger, 1985] the agent cannot select an arbitrary possible world but must instead choose one among several distributions $P_d(W)$ over all possible worlds. This choice of distribution is not arbitrary, but each decision d corresponds to one fixed distribution $P_d(W)$. This distribution is used to sample the world that defines the agents reward [Osborne and Rubinstein, 1994; Jensen, 2001]. This distribution motivates the definition of the **expected utility** of an decision d, which is the expected average outcome

$$\mathbb{E}[u|d] = \mathrm{EU}(d) = \sum_{w \in \Omega_W} u(w) P_d(w) \ .$$

In this setting the rational decision is to select the one which maximizes the expected utility.

[2] We assume unsupervised learning

Example 2.2 *Let us assume the agent can decide whether to take an umbrella for a two-day trip or not. The umbrella protects against rain. The agent's utility is defined as the sum of the costs of* 1 *(utility* $= -1$*) for taking an umbrella, and the utility of* 2 *(utility* $= 2$*) for getting wet. Let us furthermore assume that his previous estimate of the probability of switching weather conditions* $p = 0.2$ *was correct. When starting on a sunny day the probability of rain at any of the following two days is* $P(rain) = 0.2 + 0.8 \cdot 0.2 = 0.36$*. Therefore the expected utilities are* $\mathrm{EU}(no\ umbrella) = -0.72$ *and* $\mathrm{EU}(umbrella) = -1$*. Thus, the rational decision is to not take the umbrella.*

2.1.2 Directed Graphical Models

As it is typically quite cumbersome to list the probabilities of all possible outcomes, we will now introduce directed graphical models.

Directed graphical models provide a framework to define probability distributions compactly. Together with undirected models, they have become one of the most popular representations in statistical machine learning. Directed models are traditionally the preferred representation in AI and statistics, whereas undirected models are traditionally used within the physics and computer vision communities.[3] Relevant for this thesis are only directed graphical models, more precisely Bayesian networks [Pearl, 1988], which we will now introduce.

Especially in cases where it can be assumed that there is a generative process underlying the domain to be modeled, graphical models are well-suited. This generative process can be used to factor the probability distribution into a set of random variables, where a graph structure is used to encode the dependencies.

Definition 2.1 (Bayesian Network) *A* **Bayesian network** *consists of*

- *a set of random variables* $\mathbf{X} = \{X_1, \ldots, X_n\}$*,*
- *a directed graph* $G = \langle \mathbf{X}, \mathbf{E} \rangle$ *where the set of random variables* $\mathbf{X}$ *defines the vertecis and* $\mathbf{E}$ *the edges. A a node* X_j *is a parent of a node* X_i *if* $(X_j, X_i) \in \mathbf{E}$*. The set of all parents of a node* X_i *is denoted by* $pa(X_i)$*.*
- *A conditional probability distribution* $P(X_i|pa(X_i))$ *for each node* $X_i \in \mathbf{X}$*.*

The **Markov Blanket** $MB(X)$ *of a node* X *refers to the set of its parents, its children and its childrens' other parents. Every node* A *is conditionally independent of every other node* B *given its Markov blanket that is* $P(A|MB(A), B) =$

[3] Kevin Murphy. A brief introduction to graphical models and Bayesian networks. Available on web, at `http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html`, 1998, retrieved 5. May 2011.

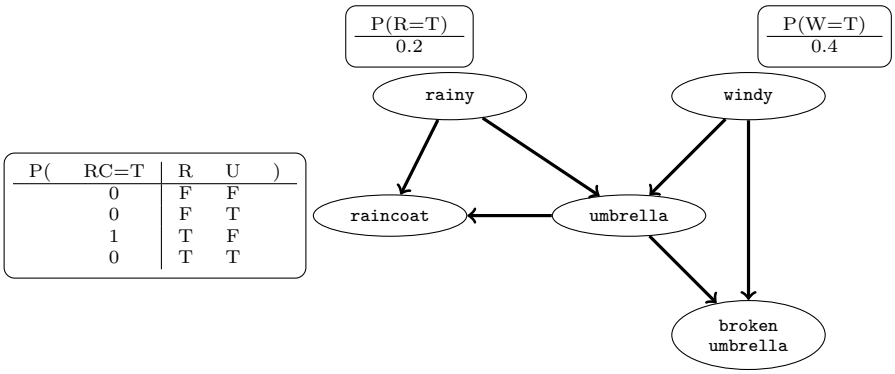


Figure 2.1: A Bayesian network encoding the weather and whether an umbrella or a raincoat is used. At windy days an umbrella is likely to break.

$P(A|MB(A))$.
If $P(\mathbf{X})$ and a graph G encode the same independence relations then the joint distribution can be factorized as follows

$$P(\mathbf{X}) = \prod_{X_i \in \mathbf{X}} P(X_i|pa(X_i)).$$

This formula is commonly referred to as the **chain rule** *for Bayesian networks.*

Without independence assumptions, an exponential number of parameters would be required to represent the joint probability distribution, whereas the number of parameters in the Bayesian network is linear in the maximal number of parents of a node. The edges in a Bayesian network are often interpreted in a causal [Pearl, 2009] way. While this assumption does not necessarily holds, a causal model typically has a very sparse graph structure.

Example 2.3 *Consider the Bayesian network in Figure 2.2. The propositions* `rainy` *and* `windy` *influence whether an umbrella is taken. At days where it is* `rainy` *but an* `umbrella` *is not taken it is more likely to take a raincoat. The Bayesian network has only $2 \cdot 2 + 3 \cdot 2^2 = 16$ parameters, whereas the same probability distribution encoded as joint probability table would have $2^5 - 1 = 31$ parameters*

Typical inference tasks in Bayesian networks are

Probability of Evidence Calculate the probability of evidence.

Most Probable Explanation Calculate the most likely instantiation of the network variables, given some evidence.

Maximum a Posteriori Hypothesis Calculate the most likely instantiation a subset of the network variables, given some evidence.

Several algorithms exist to solve these tasks for a Bayesian network. Among them are exact algorithms like, for example, variable elimination[Zhang and Poole, 1994], the jointree algorithm [Lauritzen and Spiegelhalter, 1988], and weighted model counting [Chavira and Darwiche, 2008], but also approximate ones based on belief propagation [Pearl, 1982] and Gibbs sampling [Geman and Geman, 1984]

While Bayesian networks typically provide a more-compact representation of probability distributions, due to the factorization, they are restricted to a finite set of random variables. In the following we will introduce two representations that generalize Bayesian networks towards an arbitrary number of random variables.

2.1.3 Template-Based Models

In the following we will review two kinds of graphical models which allow one to specify distributions over a potentially (countably) infinite number of random variables. The first class consists of temporal models, and the second one of plate models.

Many real-world applications require one to model time series data. Such models allow one, for example, to predict the future state given the past observations. Examples are weather forecasting or predicting how the social network of a person evolves over time. More formally: a (discrete-time) **stochastic process** defines a distribution $P(X_1, \ldots, X_T)$ over a sequence of random variables $X_1, \ldots, X_T$ that characterizes the state of the world at time $t = 1, \ldots, T$.
A stochastic process is called n-th order **Markov** if $P(X_{t+1} \mid X_t, \ldots, X_0) = P(X_{t+1} \mid X_t, \ldots, X_{t-n+1})$. The typical case where $m = 1$ are so-called Markov processes. The assumption underlying Markov processes is that the future states are independent of the past states given the current state.
A stochastic process is called **stationary** if $P(X_{t+1} \mid X_t) = P(X_{t'+1} \mid X_{t'})$ for all t, t'. Stationary Markov processes are the simplest and most widely used class of stochastic processes. We will use the notation $X_{[0:t]}$ and $X_0, \ldots, X_t$ interchangeably.
The simplest type of stochastic processes are **Markov chains**. A Markov chain assumes that each state is atomic, which means that the state cannot be factorized any further. As the assumption that all states are atomic is very restrictive, Markov chains have been extended towards dynamic Bayesian networks. This extension is similar to the extension of a single random variables towards Bayesian networks. It is often possible to factorize the initial state and transition distribution of a Markov chain into separate random variables defining a joint state.

A dynamic[4] Bayesian network (DBN) [Russell and Norvig, 2003] defines a stochastic process over a sequence of vectors of random variables $\mathbf{X}_1, \ldots, \mathbf{X}_T$. The idea underlying DBNs is to define the initial state and the transition distributions as Bayesian networks. The transition distribution is then given by a so-called **2-time-slice Bayesian network** (2TSBN). This 2TSBN models a conditional distribution $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$.

Hence it can be defined in the following way:

Definition 2.2 (Dynamic Bayesian network) *A dynamic Bayesian network is a pair $\langle B_0, B_{\rightarrow}\rangle$, which defines a probability distribution over sequences of vectors of random variables, where*

1. *The Bayesian network B_0 defines a probability distribution over the variables in $\mathbf{X}_0$, the so-called initial state distribution*
2. *The Bayesian network $B_{\rightarrow}$ defines a conditional distribution $P(\mathbf{X}_{t+1}|\mathbf{X}_t)$*

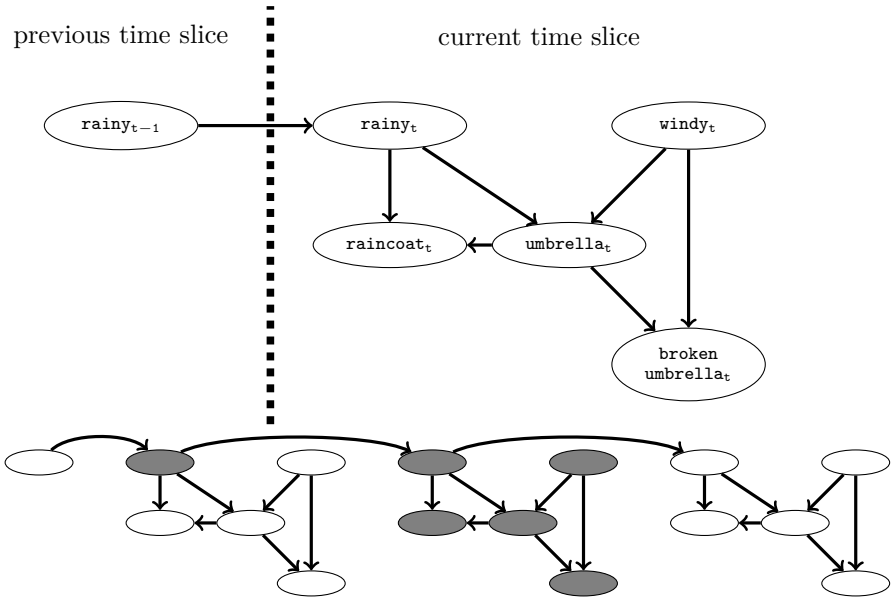


Figure 2.2: The transition distribution of a dynamic Bayesian network represented as a 2-time-slice network (top). The network unrolled over three time-slices. The leftmost node represents the prior distribution over rainy (bottom).

[4]The term "dynamic" can be misleading. Often DBNs are used to model stationary stochastic processes.

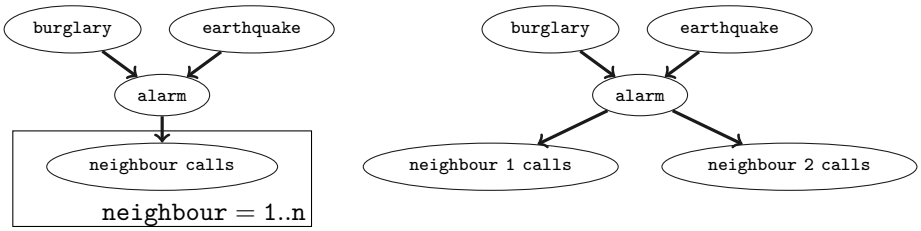


Figure 2.3: The well-known alarm network [Pearl, 1988] in plate notion (left). The instantiation of the alarm network for two persons (right).

The Bayesian network $B_{\rightarrow}$ can be seen as a template that is copied over and over again (Figure 2.2), or in other words the Bayesian network is unrolled over time. This example encodes a model which represents sequences of days instead of a single day. Whether the `coat` or the `umbrella` is taken and whether the umbrella `breaks` or it is `windy` is assumed to be independent of the previous day. The probability of `rainy`, on the other hand, depends on the values at the previous day.

While this definition allows one to define stochastic processes, with factored probability distributions, it requires that each state is defined in terms of the same finite set of random variables.

A class of models, which alleviates this restriction, are **plate model**s. There is no widely-accepted formal definition of plate models, therefore we will just give the intuition and provide an example. A plate model is an abstract description of a Bayesian network. It allows one to define arbitrarily-sized Bayesian networks, where sets of random variables are duplicated multiple times in the network. The graphical representation can be read almost like a normal Bayesian network. In difference to Bayesian networks random variables are grouped in plates, where a plate indicates the repetition of the enclosed substructure. Each repetition is assumed to be independent of the other ones. The number in the plate indicates how often the substructure is repeated. It must be possible to determine the number of repetitions either by the evidence or the number of repetitions needs to be specified as distributions.

Example 2.4 (Plate Model) *Figure 2.3 (right) depicts the well-known alarm network [Pearl, 1988]. The probability that a neighbour calls is conditionally independent of earthquake given a known value for alarm. Figure 2.3 (left) shows the same network in plate notation. This model represents the distribution for all possible numbers of neighbours.*

While this model allows one to use an arbitrary number of objects, the assumed independence prohibits the modelling of sequential models. While one can imagine

a "2TS-Plate Model" we will later give a more concise model that is based on relational modeling techniques.

2.1.4 Influence Diagrams

One of the main reasons to build statistical models like Bayesian networks is that they support a decision making process. While Bayesian networks allow one to calculate the probability of a certain event, they do not allow one to model the entire decision making process. This is because they lack the ability to model decisions and the utility of certain outcomes. Influence diagrams [Howard and Matheson, 1984/2005; Jensen, 2001] extend Bayesian networks by **utilities** and **decisions** nodes.

Definition 2.3 (Influence Diagram) *An* **influence diagram** *consists of*

- *a set of random variables* $\mathbf{X}$*,*
- *a set of decision nodes* $\mathbf{D}$ *(depicted by a square),*
- *a set of utility nodes* $\mathbf{U}$ *(depicted by a diamond),*
- *a directed acyclic graph over* $\mathbf{X} \cup \mathbf{D} \cup \mathbf{U}$*, such that no node of* $\mathbf{U}$ *is the parent of any other node*
- *and finally for each node* $X \in \mathbf{X}$ *a conditional probability distribution* $P(X \mid pa(X))$*, for each utility node* $U \in \mathbf{U}$ *a real-valued function* $f(pa(U))$ *with arguments* $pa(U)$*.*

The chain rule for influence diagrams is

$$P(X|D) = \prod_{X \in \mathbf{X}} p(X|pa(X)) \qquad \text{where } pa(X) \subseteq X \cup D$$

The interpretation of the edges in an influence diagram depends on the node type at the head of the link. Edges pointing to a probabilistic node encode probabilistic dependencies like in a Bayesian network. Edges pointing to a decision represent information. This means that all information must be known prior to taking the decision. Edges whose heads are utility nodes indicate that the utility depends on the values of the variable at its tail.

Example 2.5 *The weather model is represented as influence diagram in Figure 2.4. The nodes for umbrella and raincoat are decisions. Whether an umbrella is taken depends on the value of windy and rainy. Whether a rain coat is taken depends*

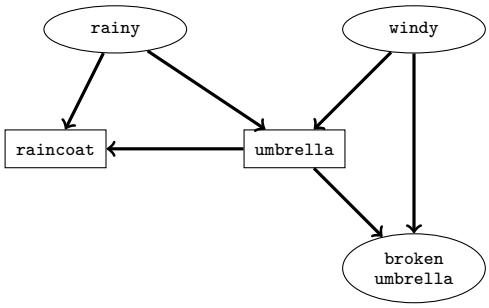


Figure 2.4: An influence diagram encoding the weather and the choices to of takeing an umbrella or a raincoat. At windy days an umbrella is likely to break.

on umbrella and rainy. Two nodes in the influence diagram are entirely new; one, which represents the costs for a broken umbrella, and the other one, which represents the costs for having neither an umbrella nor a raincoat while it is rainy.

As outlined before the typical inference task in decision problems is to maximize the expected utility. This can be done using standard Bayesian network inference methods by maximizing

$$\mathbb{E}[U|d] = \sum_{U \in \mathbf{U}} \sum_{x \in domain(pa(X))} f(x) P(X = x \mid D).$$

2.2 Binary and Algebraic Decision Diagrams

In this section we will review decision diagrams, which are data structures that will be used in large parts of this thesis. We use two variants in this thesis, namely Reduced Ordered Binary Decision Diagrams (RoBDD) and Reduced Ordered Algebraic Decision Diagrams (ADDs). In this section, we will make the distinction between RoBDD and BDD, afterwards we will use BDD and ADD as synonyms for the reduced variant.

Especially RoBDDs are popular in several fields of computer science. They represent Boolean formulas but can also be considered a compact representations of a set of sets. RoBDDs became popular after Bryant [1986] demonstrated that they can be used as a canonical representation of Boolean functions. Therefore the traditional application fields for BDDs is in hardware and software verification. But as they compress the formula, they are also used in hardware architecture. In recent years they became popular in artificial intelligence and machine learning. Examples are by Chavira and Darwiche [2007] and by Minato et al. [2007] in the context of graphical models. A more recent application is in probabilistic programming,

where they allowe efficient exact inference [Kimmig et al., 2008]. Also ADDs gained attention in AI where they have been used in solving POMDPs [Boutilier and Poole, 1996; Hansen and Feng, 2000; Sanner and Boutilier, 2009].

Let us now introduce these diagrams more formally. A Boolean formula is built from a set of literals $l_1, \ldots, l_n$ and the connectors $\wedge$ (and), $\vee$ (or) and $\neg$ (not, $\neg l$ also abbrev. as $\bar{l}$). We will use $\varphi[l \mapsto v]$ to denote that l is assigned the value $v \in \{true, false\}$. The basic idea underlying the graph representation of a formula φ, is the **Shannon decomposition** or **Shannon expansion** [Shannon, 1948].

$$\varphi = (l \wedge \varphi[l \mapsto true]) \vee (\neg l \wedge \varphi[l \mapsto false]) \quad ,$$

which generates a tree representation. More generally a Boolean formula can be represented by a rooted, directed, acyclic graph, in which nodes are annotated with variables and have out-degree 2. Each node represents a Boolean function φ. The two children are called the high-child and the low-child. The former represents the positive co-factor $\varphi[l \mapsto true]$ and the co-factor $\varphi[l \mapsto false]$ is represented by the latter. The leaves represent the logical true ($\mathbb{1}$) and false ($\mathbb{0}$). Each path in the diagram corresponds to a (partial) value assignment, which satisfies the formula if it ends in the $\mathbb{1}$-terminal and violates it otherwise.

Example 2.6 *Consider the Boolean function $\neg a \wedge ((b \wedge c) \vee (\neg b \wedge c))$. The Shannon decomposition of this function is depicted as a graph in Figure 2.5. The high-child is indicated by the solid edge and the low child by the dashed edge. For example, the left-most branch corresponds to the assignment $a = b = c = true$, which makes the formula false as the $\mathbb{0}$-terminal indicates.*

As the example shows decomposing the formula introduces a lot of redundancy. Therefore Reduced Ordered BDDs apply two reduction techniques, which are

Merging isomorphic sub-graphs: If two sub-graphs represent the same function they are isomorphic. Hence one can be removed by redirecting, after all incoming edges to the other one.

Node removal: Nodes with two identical children can be removed. All inking edges are redirected to either of these children.

These reductions are repeatedly applied until no further reductions are possible. Given a good order for performing the Shannon decompositions often results in BDDs that are very small.

Example 2.7 *Repetitively applying the reduction to the BDD in Figure 2.5 (left) yields a RoBDD (middle). Examples where node merging can be applied are the leaves and the areas shaded. Examples for node removal are the two leftmost c nodes.*

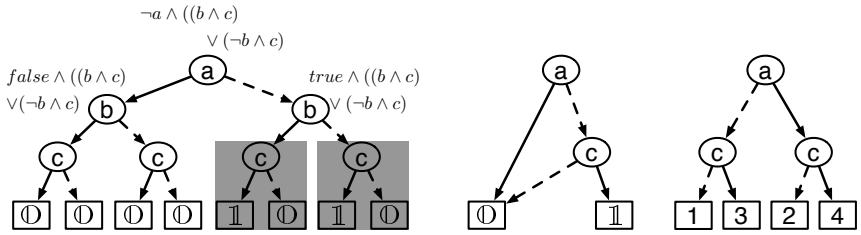


Figure 2.5: An ordered Binary Decision Diagram before reducing it (left). The two top levels are annotated with the functions represented. The shaded areas represent isomorphic sub-trees. The two leftmost arrows both point to the same node, hence the c node can be removed. The resulting Reduced Ordered Binary Decision Diagram is more compact (middle). Also real-valued functions can be represented using binary trees, which are called ADDs (right)

In practice BDDs are not built by starting from a complete tree, which is compressed afterwards, but built bottom-up. The bottom-up construction uses the standard Boolean connector as operations on BDDs. While negating a BDD is a constant time operation and forming the conjunction or disjunction of two BDDs are polynomial time operations, generating an entire BDD may be exponential in the size of the formula. This is because the size of the resulting BDD can be the product of the size of the original BDDs. In fact, it might grow exponentially, during the bottom-up construction. The size can be reduced by adjusting the order of the variables in the BDD. However, finding the optimal order, which minimizes the size is a co-NP problem [Bryant, 1986]. Therefore, state-of-the-art BDD implementations use heuristic methods, which often suffice in practice.

The second class of Decision Diagrams used in this thesis are Algebraic Decision Diagrams. ADDs are popular as they can compactly represent very large matrices. **ADD**s belong to the family of **Multi-Terminal BDD**s [Clarke et al., 1997]. Multi-Terminal BDDs generalize BDDs such that leaves may take more values than true and false. In an ADD the domain of the terminals is $\mathbb{R}$. Therefore ADDs represent functions $f : [0, 1] \to \mathbb{R}$. Using the Shannon decomposition, the function can again be split into its co-factors. The only difference to BDDs is that each node now represents a real value.

Example 2.8 *Consider the* 4×4 *matrix*

$$M = \left(\begin{array}{c|cccc} & \overline{a}\,\overline{b} & \overline{a}b & a\overline{b} & ab \\ \hline \overline{c}\,\overline{d} & 1 & 1 & 2 & 2 \\ \overline{c}d & 1 & 1 & 2 & 2 \\ c\overline{d} & 3 & 3 & 4 & 4 \\ cd & 3 & 3 & 4 & 4 \end{array}\right).$$

If the columns are identified by (a, b) and the rows by (c, d) as indicated, then this matrix can be represented using the Boolean function $(a, b, c, d) = a + 2 \cdot c + 1$. This function and therefore the matrix can be compactly described using the ADD in Figure 2.5 (right).

As such they are usually also not constructed bottom up, but again by using algebraic functions like the scalar multiplication $c \cdot g$ of an ADD g with the constant c, the addition $f \oplus g$ of two ADDs, and the *if-then-else* test $\text{ITE}(b, f, g)$ which represents the Shannon decomposition.

2.3 Logic Programming

This section briefly[5] reviews the concepts of logic programming. Logic programming provides a declarative language, which is the basis of the representations used throughout this thesis.

An alphabet Σ is a set of predicates, constants and variables. A **logical atom** is an expression $p(t_1, \ldots, t_n)$ where p is a **predicate** symbol of arity n written as p/n. The t_i are **terms**, built from constants, variables and structured terms. **Constants** are denoted by lower case symbols and **variables** by upper case symbols. Structured terms are of the form $f(t_1, \ldots, t_k)$, where f is a functor symbol and the t_i are terms. We use $var(a)$ to denote all variables of the atom a. An expression is **ground** if it does not contain any variables. Ground atoms are frequently called **facts**. The set $HB(\Sigma)$ of all ground atoms built using the alphabet Σ is called the **Herbrand base** of Σ.[6] A **Herbrand interpretation** I is a subset of the **Herbrand Base** $HB(\Sigma)$ that lists all true facts.

Example 2.9 (Relational weather domain) *Consider the weather domain of example 2.3. A first-order alphabet of this domain looks as follows. The first day is represented using the constant term* `0`*. The functor* `s/1` *maps a day onto its successor day. Therefore, the term* `s(0)` *represents the second day. Furthermore we use the predicate* `weather/2`*, and two constants:* `rainy`*,* `sunny`*. Rainy weather on day* `s(0)` *is represented by the ground atom* `weather(s(0),rainy)`*. The constants* `windy`*,* `calm` *and the predicate* `wind/2` *refer to the wind condition. Finally,* `umbrella/1` *indicates whether an umbrella was taken. The Herbrand-interpretation*

$$\{\texttt{weather(s(0),rainy)}, \texttt{wind(s(0),windy)}\}$$

represents, for example, the weather on the first day.

[5] A more detailed study can be found in [Flach, 1994], [Nilsson and Małuszyński, 1990], and [Bratko, 1990]

[6] If the program does not contain any constant, an arbitrary constant is added.

A variable, like X, is a term whose value is unknown. A substitution θ maps variables to terms. The substitution is a set of expressions $\mathtt{X}/\mathtt{c}$ indicating that the variable X is mapped to the constants c. Two substitutions θ, θ' can be concatenated written as $\theta \circ \theta'$. For an atom a, the two expressions $(\mathtt{a}\theta)\theta'$ and $\mathtt{a}(\theta \circ \theta')$ yield the same result The atom $a\theta$ is obtained from a by replacing all variables by terms according to θ. Two atoms a and b are called unifiable if there exists a substitution θ such that $\mathtt{a}\theta = \mathtt{b}\theta$. In that case θ is called a unifiers. A unifier θ is called the **most general unifier** of a and b written as $\theta = mgu(\mathtt{a}, \mathtt{b})$ if and only if every other unifier λ of a and b is a concatenation of θ and a third unifier.

Example 2.10 (Relational weather domain (contd.)) *Any sunny day can be referred to by the atom* $a = \mathtt{weather}(\mathtt{X}, \mathtt{sunny})$*, whereas the atom* $b = \mathtt{weather}(\mathtt{s}(\mathtt{0}), \mathtt{Y})$ *refers to the first day. Substitutions unifying* a *and* b *are*

$$\{\mathtt{Y}/\mathtt{sunny}, \mathtt{X}/\mathtt{s}(\mathtt{0})\}, \qquad \{\mathtt{X}/\mathtt{s}(\mathtt{0}), \mathtt{Y}/\mathtt{sunny}\},$$

$$\{\mathtt{X}/\mathtt{s}(\mathtt{0}), \mathtt{Y}/\mathtt{Z}, \mathtt{Z}/\mathtt{sunny}\}, \qquad \{\mathtt{X}/\mathtt{s}(\mathtt{Z}), \mathtt{Z}/\mathtt{0}, \mathtt{Y}/\mathtt{rainy}\},$$

but only the first two are most general unifier.

A **definite clause** (or clause) c is an expression of the form $\mathtt{h} :- \mathtt{b_1}, \ldots \mathtt{b_n}.$ where h and the $\mathtt{b_i}$ are logical atoms. We use $head(c)$ to refer to h and $body(c)$ to refer to $\mathtt{b_1}, \ldots, \mathtt{b_n}$. A set of definite clauses P is called a program. If the only atom in the body of a definite clause is true the clause is a **fact** (also called a **unit-clause**) and then body can be omitted. A clause is called **range restricted** if all variables occurring in the head also occurs in the body. The intuitive meaning of such a clause is that "whenever all atoms in the body are true for some substitution the head after the substitution has been applied is true as well".

A **logic program** is a set of definite clauses and facts. If a program P is constructed using the symbols of the alphabet Σ we define the Herbrand base of P as $HB(P) = HB(\Sigma)$.

Example 2.11 (Weather Logic Program) *Using the symbols of the weather domain the following logic program* P_W *can be constructed*

$\mathtt{weather}(\mathtt{0}, \mathtt{sunny}).$

$\mathtt{weather}(\mathtt{s}(\mathtt{Day}), \mathtt{Y}) :- \mathtt{weather}(\mathtt{Day}, \mathtt{Y}).$

$\mathtt{wind}(\mathtt{Day}, \mathtt{windy}) :- \mathtt{weather}(\mathtt{Day}, \mathtt{rainy}).$

$\mathtt{wind}(\mathtt{Day}, \mathtt{calm}) :- \mathtt{weather}(\mathtt{Day}, \mathtt{sunny}).$

The first line, a fact, states that the first day is a sunny day. The second line, a clause, states whatever weather it is, it will be the same weather the next day. The last two lines state that `rainy` *days are* `windy` *and* `sunny` *days are* `calm`*.*

The semantics of a definite clause program is given by its least Herbrand model. An interpretation I is a model of a program P if for all $h :- b_1, \ldots, b_n$ and substitutions θ, it holds that if $\{b_1\theta, \ldots, b_n\theta\} \in I$ then $h\theta \in I$. The least Herbrand model $LH(P)$ is the minimal such set. A logic program P entails a fact f written as $P \models f$, if f is in each model of P.

We will now show how to compute the *least Herbrand model*, which defines the semantics of a logic program.[7] The **least Herbrand model** can be computed as the fixed-point of the immediate consequence operator:

Definition 2.4 (T_p operator) *For a definite clause program P the immediate consequence operator T_P applied to a set of facts I is defined as*

$$T_P(I) = \{h\theta \mid (h :- b_1, \ldots, b_n) \in P \text{ and } \exists\theta \text{ s.t. } h\theta \in HB(P) \text{ and } b_i\theta \in I\}$$

It follows immediately from the definition that every fixed-point of the T_P operator is a model of P. If the least Herbrand model is finite, the fixed-point of a logic program can be calculated in the following way: we define $I_0 = \varnothing$ and recursively generate $I_{m+1} = T_P(I_m)$; whenever this procedure reaches a fixed-point, that is $I_m = T_P(I_m)$, then I_m is the least fixed-point of P.

Example 2.12 (Fixed point of the weather program) *The interpretations constructed by repeatedly applying the T_{P_W} are*

$$\begin{aligned}
I_0 = &\ \{\} \qquad\qquad\qquad\qquad I_1 = \{\texttt{weather(0, sunny)}\} \\
I_1 = &\ \{\texttt{weather(0, sunny)}, \texttt{weather(s(0), sunny)}, \texttt{wind(0, calm)}\} \\
I_2 = &\ \{\texttt{weather(0, sunny)}, \texttt{weather(s(0), sunny)}, \texttt{wind(0, calm)}, \\
&\ \texttt{weather(s(s(0)), sunny)}, \texttt{wind(s(0), calm)} \qquad\qquad\qquad\qquad \} \\
&\ \vdots
\end{aligned}$$

2.3.1 Inference in Logic Programs

The typical inference task in logic programming is to decide whether a **query** given as conjunction of atoms, $? - \mathtt{q_1}, \ldots, \mathtt{q_m}$, is entailed by a program. While entailment can be tested by first computing the least fixed-point, this is impractical. Instead most logic programming systems use **selective linear definite clause**

[7] There are multiple possible semantics for logic programs. Luckily they coincide for stratified, negation free logic programs.

(SLD) resolution.[8] SLD-resolution is a **refutation** process, where the negation of the query is added to the program and resolution is used to derive the empty clause.

The inference rule applied to refute a query ?-$\mathtt{q}_1, \ldots, \mathtt{q}_m$. is to select a clause $\mathtt{h} :- \mathtt{b}_1, \ldots, \mathtt{b}_n$ such that $\theta = mgu(\mathtt{h}, \mathtt{q}_1)$ exists and compute the resolvent $\mathtt{b}_1\theta, \ldots, \mathtt{b}_n\theta, \mathtt{q}_2, \ldots, \mathtt{q}_m$. If some of the variables in $\mathtt{b}_i$ also occur in any of the $\mathtt{q}_i$ they have to be renamed. If any sequence of resolution steps exists that ends in the empty query ?-. typically indicated by $\square$, the goal is proven. If the goal can be proven by SLD-resolution in a cycle free program, then the query is entailed by the Least Herbrand model. A program is called cycle free, if the SLD-derivation of an atom a does not contain a.

Example 2.13 (SLD derivation) *A possible derivation of* $\mathtt{wind}(\mathtt{0}, \mathtt{X})$ *in the weather program is*

query	*clause*	*substitution*
$\mathtt{wind}(\mathtt{s}(\mathtt{X}), \mathtt{calm})$	$\mathtt{wind}(\mathtt{Day}, \mathtt{calm}) :- \mathtt{weather}(\mathtt{Day}, \mathtt{sunny}))$	$\{\mathtt{Day}/\mathtt{s}(\mathtt{X})\}$
$\mathtt{weather}(\mathtt{s}(\mathtt{X}), \mathtt{sunny})$	$\mathtt{weather}(\mathtt{s}(\mathtt{Y}), \mathtt{sunny}) :- \mathtt{weather}(\mathtt{Y}, \mathtt{sunny})$	$\{\mathtt{Y}/\mathtt{X}\}$
$\mathtt{weather}(\mathtt{X}, \mathtt{sunny})$	$\mathtt{weather}(\mathtt{0}, \mathtt{sunny})$	$\{\mathtt{X}/\mathtt{0}\}$

Note that this procedure is non-deterministic, as there might be several clauses which can resolve the current sub-goal. Therefore, in implementations of SLD resolution, e.g., Prolog the clauses are tried in the order in which they are listed. Whenever a derivation fails, this means no clause head is unifiable with $\mathtt{q}_i$ and the algorithm backtracks to the last choice of a clause. SLD-resolution continues this process until the empty clause is derived or no rule can be unified with $\mathtt{q}_1$. The former case means the goal is proven. In the latter case, the algorithm backtracks to the last choice. We will later see another example, where the resolution process is also depicted graphically (cf. Example 3.3 and Figure 3.1).

An alternative procedure to test whether a query is entailed, is based on Clark's completion. While it was developed to assign semantics to logic programs with negation, it can also be used to perform inference. The completion of a program is a propositional formula in conjunctive normal form, typically compactly written using equivalences.

For a ground[9] logic program P the completion of P of with respect to the atom $\mathtt{h}$ is

$$\mathtt{h} \longleftrightarrow (\mathtt{body}_1 \vee \ldots \vee \mathtt{body}_n).$$

[8] In fact, SLD resolution calculates the answer substitutions for a query. This is the set of all substitutions θ such that $\mathtt{q}_1\theta, \ldots, \mathtt{q}_n\theta$ is entailed by the program. As for this thesis only entailment is relevant, we will omit the details on the substitutions.

[9] For the non-ground case see Nilsson and Małuszyński [1990]

where the `body`$_i$'s are the bodies of all clauses having `h` as head. The completion $comp(P)$ of a logic program is the conjunction of the completion of all atoms. This is

$$\bigvee_{\exists h \,:\, (h:\!-body)\in P} \left[h \longleftrightarrow \bigwedge_{h:\!-body_i \in P} body_i \right] .$$

The conjunction of all such clauses is equivalent to the original logic program for all acyclic programs.[10] This means that if a fact is entailed by a logic program it is also entailed by its completion.

[10]To be more precise for all tight programs. That is a program which does not contain any positive cycle.

Foundations: Probabilistic Logic Programming

3

This chapter reviews probabilistic logic programming. We present the distribution semantics, which provides the theoretical foundations for most probabilistic logic programming languages. Afterwards we introduce two languages, Problog and CP-Logic, together with their typical inference algorithms.

The previous chapter introduced probabilistic graphical models and logic programming. This chapter shows how these can be combined leading to the notion of probabilistic logic programming. It combines the expressiveness and intuitive semantics of logic programming with the ability of graphical models to represent uncertainty.

Probabilistic programming languages augment programming languages with stochastic choices so that they define a probability distribution over programs. Probabilistic logic programming languages, which are the basis for the models and algorithms developed throughout this dissertation, are based on logic programming.

The theoretical foundation for most probabilistic programming languages is provided by the *distribution semantics*. However, the distribution semantics only proves the existence of a probability distribution, but does not describe how to calculate

the probability of a query.[1] Therefore, we discuss two concrete languages together with the typical algorithms for computing the probability of a query.

The first language is ProbLog [De Raedt et al., 2007], which implements the distribution semantics in a direct way. This makes ProbLog ideal for theoretical and algorithmic analyses. The second language is CP-Logic whose syntax is based on causal processes. Therefore, the resulting models are often more intuitive, but at the expense that theoretical analysis can be more difficult. Furthermore, developing algorithms seems to be more complicated.

3.1 Distribution Semantics

A wide variety of formalisms that augment logic programming languages with probabilistic concepts have been developed. Popular approaches are PRISM [Sato and Kameya, 1997], ICL [Poole, 1997], ProbLog [De Raedt et al., 2007], CP-Logic [Vennekens et al., 2006]. All of these formalisms can be considered instances of the **distribution semantics** [Sato, 1995]. The distribution semantics is rigorously defined and shows that all subsumed languages define a unique probability distribution over interpretations and queries. Its main feature is that it allows for countably many random variables in the form of **probabilistic facts** or probabilistic choices. The distributions semantics then specifies a distribution over interpretations, called possible worlds. The distribution semantics does not only provide a semantics for probabilistic logic programming languages is also relevant for many other formalisms like, for example, Markov logic or Church. Markov logic [Domingos and Lowd, 2009] also represents distributions over interpretation. It can be represented terms of the distribution semantics, by mapping it on ProbLog (see [Gutmann, 2011]). Church [Goodman et al., 2008] can easily be mapped on CP-Logic by using the standard technique of mapping an n-ary function to an $n+1$-ary relation. This trick allows one to map each function definition to a clause. The function call is represented by the head of the clause and the function body by the body of the clause. Finite discrete elementary random procedures can be represented using annotated disjunctions like in CP-Logic. Mapping the representation of continuous distribution from Church to the distributions semantics is a bit more elaborate, but can be achieved along the lines of distributional clauses [Gutmann et al., 2011b], which are an extension of CP-Logic towards continuous distributions.

The distribution semantics is a generalization of the least Herbrand model semantics. The basic idea is that the facts of a logic program are not certainly true or false. Instead, the truth value of a single fact or set of facts is governed by a probability distribution. Once the truth value of the probabilistic facts is sampled, the usual

[1] In Sato [1995] describe also the PRISM system, which restricts the logic program further.

Herbrand model semantics applies to the resulting logic program. Hence, there is no longer a unique least Herbrand model. Instead, the model depends on the truth values chosen for the individual probabilistic facts. In essence, a probabilistic logic program defines a distribution over least Herbrand models by means of a distribution over probabilistic facts. This distribution over Herbrand models defines the **success probability** of first-order queries, which corresponds to the probability that the query is true in a randomly sampled Herbrand model.

More formally, a **probabilistic logic program** is a tuple $\langle \mathcal{F} \cup \mathcal{BK}, (P_{\mathcal{F}}^{(n)})_{n \in \mathbb{N}} \rangle$, where $\mathcal{BK}$ is a set of ground definite clauses and $\mathcal{F}$ a set of ground facts. The logic program $\mathcal{T} = \mathcal{BK} \cup \mathcal{F}$ is such that it uses only countably many variables, functors and constants. If the logic program of interest is not ground, it can automatically be grounded. The predicates in $\mathcal{F}$ and the heads of the clauses in $\mathcal{BK}$ have to be disjoint. Finally $P_{\mathcal{F}}^{(n)}$ defines a series of distributions, each over a (finite) subset of the facts $\{A_1, \ldots, A_n\} \in \mathcal{F}$. It is assumed that $\{A_1, \ldots, A_n\}$ are the same facts for all $P_{\mathcal{F}}^{(m)}$, where $m > n$.

While the distribution semantics restricts $\mathcal{BK}$ to non unit-clauses, we will relax this restriction. A unit clause is treated like a probabilistic fact with probability 1. The use of definite clauses prohibits the use of negation in the body of a clause. As this is often too restrictive, we allow for negation on ground probabilistic facts. The negation of a fact f is written as $not(f)$, where $P_{\mathcal{F}}^{(n)}$ needs to be 0, when both are true or both are false. This can be done automatically in the following way. Replace $\mathcal{F}$ by

$$\mathcal{F}' = \{f \text{ and } not(f) | f \in \mathcal{F}\}$$

The distribution over the facts needs to be defined as $P_{\mathcal{F}'}^{(2n)}(A_1 = x_1, not(A_1) = \neg x_1, \ldots, A_n = x_n, not(A_n) = \neg x_n) = P_{\mathcal{F}}^{(n)}(A_1 = x_1, \ldots, A_n = x_n))$ and $P_{\mathcal{F}'}^{(2n)}(\ldots, A_i = x_i, not(A_i) = x_i, \ldots) = 0$.

To be able to characterize the distribution defined by a probabilistic logic program, we need to introduce the following two properties:

- **Finite support condition:** the truth value of all head atoms of clauses in $\mathcal{BK}$ is determined by finitely many facts. If the finite support condition is fulfilled, each sequence of resolution steps ends after a finite number of steps, and only finitely many clauses share the same head element.

- **Kolmogorov compatibility condition:** the distributions $P_{\mathcal{F}}^{(n)}$ fulfill

$$\sum_{x \in \{true, false\}} P_{\mathcal{F}}^{(n)}(A_1 = x_1, \ldots, A_{n-1} = x_{n-1}, A_n = x) =$$

$$P_{\mathcal{F}}^{(n-1)}(A_1 = x_1, \ldots, A_{n-1} = x_{n-1}),$$

  which means that marginalizing out A_n in $P_{\mathcal{F}}^{(n)}$ yields $P_{\mathcal{F}}^{(n-1)}$.

If a program fulfills the finite support and the Kolmogorov compatibility condition, then there exists a unique distribution $P_{\mathcal{T}}$ over Herbrand interpretations $I \in HB(\mathcal{T})$, where $P_{\mathcal{T}}(LH(F_\omega \cup \mathcal{BK})) = P_{\mathcal{F}}(F_\omega)$. Furthermore the distribution $P(q)$ over first-order queries is well defined. The existence of these distributions can be proven by means of the Kolmogorov extension theorem [Sato, 1995].
We will now illustrate the idea of this extension process using an example. To simplify notation, we abbreviate $P_{\mathcal{F}}^{(n)}(A_1 = true, A_2 = false, \ldots, A_n = false)$ in the following as $P_{\mathcal{F}}^{(n)}(\langle 1, 0, \ldots, 0\rangle)$.

Example 3.1 *Consider a simplified version of the umbrella example 2.3. As probabilistic program $\langle \mathcal{F}, \mathcal{BK}, (P_{\mathcal{F}}^{(n)})\rangle$ it consists of*

$$\mathcal{F} = \{\texttt{rainy}, \texttt{windy}, \texttt{not(windy)}\}$$

$$\mathcal{BK} = \{\texttt{umbrella} :\!- \texttt{rainy}, \texttt{not(windy)}.\}$$

Assuming that $A_1 =$ `rainy`, $A_2 =$ `windy` and $A_3 =$ `not(windy)`. The basic distribution $P_{\mathcal{F}} = P_{\mathcal{F}}^{(3)}$ assigns each element $\omega = \{0, 1\}^3$ a probability, e.g.,

$$P_{\mathcal{F}}(\langle 0, 0, 1\rangle) = 0.48 \qquad P_{\mathcal{F}}(\langle 1, 0, 1\rangle) = 0.12$$

$$P_{\mathcal{F}}(\langle 0, 1, 0\rangle) = 0.32 \qquad P_{\mathcal{F}}(\langle 1, 1, 0\rangle) = 0.08$$

and 0 otherwise.

Each sample ω represents one interpretation $F_\omega \subseteq \mathcal{F}$ and hence defines a unique least Herbrand model $LH(F_\omega \cup \mathcal{BK})$. This implies that the distribution $P_{\mathcal{F}}$ can then be extended towards a unique distribution $P_{\mathcal{T}}$ over Herbrand interpretations where $P_{\mathcal{T}}(LH(F_\omega \cup \mathcal{BK})) = P_{\mathcal{F}}(F_\omega)$

As there are only countably many ground atoms – the Herbrand base of $\mathcal{T}$ – each model can be represented by a string $\omega_{\mathcal{T}}$ of countable length. Assuming the order of the probabilistic facts in $\mathcal{F}$ stays as before and $A_4 = umbrella$ then $P_{\mathcal{T}}(\langle 1, 0, 1, 1\rangle) = 0.12$ and $P_{\mathcal{T}}(\langle 1, 0, 1, 0\rangle) = 0.0$.

Real-world examples often require infinite models. Consider, for example, an extension of the previous model where the weather changes over time. The size of the interpretation is unbounded if the modeled sequences can be infinitely long. Therefore we define the **success probability** of formulas q over the Herbrand universe of $\mathcal{T}$. For that we need to define the **set of explanations** $[\![q]\!]$ of the query q, each **explanation** is a subsets of $\mathcal{F}$ that allows to proof the query q, that is

$$[\![q]\!] := \{\mathcal{F}_\omega \subseteq \mathcal{F} \mid \mathcal{F}_\omega \cup \mathcal{BK} \models q\}. \tag{3.1}$$

If $[\![q]\!]$ is finite the probability of the query is defined as

$$P([\![q]\!]) := \sum_{\mathcal{F}_\omega \in [\![q]\!]} P_{\mathcal{F}}(\mathcal{F}_\omega).$$

In the infinite case the sum is a series. However due to the finite support condition A_1 till A_n are sufficient. The compatibility condition implies that extending the distribution further does not change the probability. This can be shown formally by means of the Kolmogorov extension theorem.

We define $\mathcal{F}_q$ as the finite subset of $\mathcal{F}$, which contains all facts $A_1 \ldots, A_n$. Summing over all finite subsets of $\mathcal{F}_q$:

$$P([\![q]\!]) = \sum_{\substack{\mathcal{F}_\omega \subseteq \mathcal{F}_q \\ \mathcal{F}_\omega \cup \mathcal{BK} \models q}} P^{(n)}_{\mathcal{F}}(\mathcal{F}_\omega) \tag{3.2}$$

yields the probability of q. Sato [1995] also shows that the distribution semantics defines the probability of all first-order queries as

$$\lim_{n\to\infty} P_{\mathcal{T}}([\![q(t_1) \wedge \ldots \wedge q(t_n)]\!]) = P_{\mathcal{T}}([\![\forall x : \ q(x)]\!])$$

$$\lim_{n\to\infty} P_{\mathcal{T}}([\![q(t_1) \vee \ldots \vee q(t_n)]\!]) = P_{\mathcal{T}}([\![\exists x : \ q(x)]\!])$$

Example 3.2 *Extending example 3.1 such that each atom has an argument indicating the day yields:*

$$\mathcal{F} = \{\texttt{rainy(0)}, \texttt{windy(0)}, \texttt{not(windy(0))},$$
$$\texttt{rainy(1)}, \texttt{windy(1)}, \texttt{not(windy(1))}, \ldots\}$$
$$\mathcal{BK} = \{\texttt{umbrella(X)} :\!- \texttt{rainy(X)}, \texttt{not(windy(X))}.\}$$

The probability of rain on the current day depends on whether it has rained the day before.

We then have that

$$P_{\mathcal{F}}^{(3)}(\langle 0,0,1\rangle) = 0.48 \qquad P_{\mathcal{F}}^{(3)}(\langle 1,0,1\rangle) = 0.12$$

$$P_{\mathcal{F}}^{(3)}(\langle 0,1,0\rangle) = 0.32 \qquad P_{\mathcal{F}}^{(3)}(\langle 1,1,0\rangle) = 0.08$$

and

$$P_{\mathcal{F}}^{(3n)}(\langle \ldots, x_{ry}, x_{wy}, x_{ny}, x_r, x_w, x_n\rangle) = P_{\mathcal{F}}^{(3n-3)}(\langle \ldots, x_{ry}, x_{wy}, x_{ny}\rangle) \cdot P_{rain}(x_r|x_{ry}) P_{wind}(x_w) P_{nwind}(x_n|x_w)$$

where $P_{rain}(x_r|x_{ry}) = 0.8$ *if* $x_r = x_{ry}$, $P_{wind}(1) = 0.4$, *and* $P(x_w|x_{nw}) = 0$ *if* $x_w = x_{nw}$, *the probability of* `umbrella(1)` $= 0.4 \times [0.2 \cdot 0.8 + 0.8 \cdot 0.2]$.

3.2 ProbLog

The first language we consider that implements the distribution semantics is ProbLog. ProbLog assumes that all probabilistic facts are mutually independent. This allows one to define the basic distribution $P_{\mathcal{F}}$ along with $\mathcal{F}$, in terms of a set of labeled facts $\mathcal{F}_l$. A **labeled fact** $p_n :: f_n \in \mathcal{F}_l$ is annotated with the probability p_n that $f_n\theta$ is true for all substitutions θ grounding f_n. Actually, each non-ground labeled fact can be seen as a template for its labeled ground instances. The resulting ground facts $f_n\theta$ are called **atomic choices** and represent independent random variables. As before, we denote the set of ground probabilistic facts by $\mathcal{F} = \{f\theta \mid p :: f \in \mathcal{F}_l \text{ and } f\theta \text{ is ground}\}$.

Example 3.3 *The following ProbLog theory states that there is a burglary with probability* 0.1, *an earthquake with probability* 0.2 *and if either of them occurs than the alarm will go off. If the alarm goes off, a person* `X` *will be notified and will therefore call with the probability of* `al(X)`, *that is,* 0.7.

$$\mathcal{F}_l = \{\texttt{0.1} :: \texttt{burglary},\ \texttt{0.2} :: \texttt{earthquake},\ \texttt{0.7} :: \texttt{al(X)}\}$$

$$\begin{aligned}\mathcal{BK} = \{&\texttt{person(mary).} \quad \texttt{person(john).}\ ,\\ &\texttt{alarm} :\!- \texttt{burglary.}\\ &\texttt{alarm} :\!- \texttt{earthquake.}\\ &\texttt{calls(X)} :\!- \texttt{person(X), alarm, al(X).}\}\end{aligned}$$

The set of ground atomic choices is $\{\texttt{burglary}, \texttt{earthquake}, \texttt{al(john)}, \texttt{al(mary)}\}$.

The distribution semantics guarantees that each ProbLog program defines a distribution over queries. To apply this property, we need to define $P_{\mathcal{F}}^{(n)}$, which can be done recursively, exploiting the independence of the labeled fact as

$$P^{(n)}(A_1 \ldots, A_n) = \begin{cases} p_{f(A_n)} \cdot P^{(n-1)}(A_1, \ldots, A_{n-1}) & \text{if } A_n = true \\ (1 - p_{f(A_n))}) \cdot P^{(n-1)}(A_1, \ldots, A_{n-1}) & \text{otherwise} \end{cases}$$

$$= \prod_{A_i = true} p_{f(A_i)} \prod_{A_i = false} (1 - p_{f(A_i)})$$

where $f(A_n) = k$ if A_n is a ground instance of the fact $p_k :: f_k$. The probability of the empty set of atoms is $P^{(0)} = 1$.

Example 3.4 *This allows us to define the distributions $P_{\mathcal{F}}^{(n)}$ for the theory in example 3.3. Following the order in the definition of $\mathcal{F}_L$ the probabilities are $p_1 = 0.1$ (*$\texttt{burglary}$*), $p_2 = 0.2$ (*$\texttt{earthquake}$*), and $p_3 = 0.7$ (*$\texttt{al(X)}$*). Hence, $f(\texttt{burglary}) = 1$, $f(\texttt{earthquake}) = 2$, and $f(\texttt{al(john)}) = f(\texttt{al(mary)}) = 3$. This allows to calculate for instance the probability*

$$P^{(3)}(\{\texttt{burglary} = true, \texttt{earthquake} = false, \texttt{al(john)} = true\}))$$

$$= 0.1 \times 0.7 \times (1 - 0.2) = 0.056$$

3.2.1 ProbLog Inference

The success of ProbLog can be explained by the availability of efficient inference algorithms. The algorithm for exact inference proceeds in two steps. First, the query q is translated into a compressed representation of the possible worlds $[\![q]\!]$. Second, this representation is used to calculate the probability of the query.

More formally, the inference task is to calculate the probability of a query:

$$P([\![q]\!]) = \sum_{\substack{\mathcal{F}_\omega \subseteq \mathcal{F}' \\ \mathcal{F}_\omega \cup \mathcal{BK} \models q}} P_{\mathcal{F}}^{(n)}(\mathcal{F}_\omega) \ ,$$

where the query q can be a ground or non-ground atom or even a conjunction, disjunction of atoms. The set $[\![q]\!]$ corresponds to the set of all programs $\{\mathcal{F}_\omega \cup \mathcal{BK} \mid \mathcal{F}_\omega \cup \mathcal{BK} \models q\}$. As $[\![q]\!]$ might be infinite due to non-ground probabilistic labeled facts, we resort to the set of all explanations. An explanation E is a subset of $E \subseteq \mathcal{F}$, which is sufficient to prove the query q, that is $E \cup \mathcal{BK} \models q$. A set of

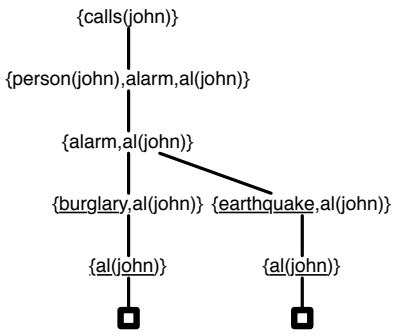


Figure 3.1: The SLD tree for `calls(john)`. Underlined facts are added to the explanation by Algorithm 1.

explanations is a set of sets, which is finite due to the *finite support condition*. Every explanation represents all those programs which contain at least the listed facts. The set of explanations can be represented by an equivalent propositional DNF. We will use the two notations interchangeably.

Example 3.5 *In the alarm network the set of explanations for the query* $\texttt{calls(john)}, \texttt{alarm}$ *is* $\{\{\texttt{burglary}, \texttt{al(john)}\}, \{\texttt{earthquake}, \texttt{al(john)}\}$. *The equivalent DNF is* $(\texttt{burglary} \wedge \texttt{al(john)}) \vee (\texttt{earthquake} \wedge \texttt{al(john)})$.

From the logic programming point of view, each explanation corresponds to (at least) one successful SLD-resolution (see Figure 3.1 for all SLD resolutions for `al(john)`). Hence, explanations can be collected automatically with an extended version of SLD resolution. While normal SLD-resolution reports that the goal is provable once the negation of the goal is refuted, the extended version given in Algorithm 1 generates all proofs and returns a set of explanations. Note that the same explanation might be generated multiple times and does not need to be minimal. This can be resolved using a post-processing step, but the algorithm to compute the probability of a set of explanations obtained by SLD-resolution will resolve this automatically.

We use the set of explanations to calculate the probability of the query. But while the calculation of the probabilities of each explanation – the product of the probability labels – is straightforward, the calculation of the overall probability is a bit more involved. This problem is known as the *disjoint-sums-of-products problem*, which we now illustrate.

Example 3.6 *In the alarm network the set of explanations for the query* $\texttt{calls(john)}, \texttt{alarm}$ *is* $\{\{\texttt{burglary}, \texttt{al(john)}\}, \{\texttt{earthquake}, \texttt{al(john)}\}\}$. *The probability of the first explanation is* 0.07 *and* 0.14 *of the second one. However the*

Algorithm 1 Generating the set of all explanations for the query q.

```
function GenExplanations(Query q, program T = F ∪ BK)
    return GenExplanationsR(q, ∅,T)

function GenExplanationsR(Query q, partial explanation E, program T)
    if q = ∅ then
        return E
    else                                              ▷ q is {q_1, ..., q_n}
        if q_1 ∈ F then
            return GenExplanationsR((q_2, ..., q_n),E ∪ {q_1},T)
        else if q_1 = not(q'_1) and q'_1 ∈ F then
            return GenExplanationsR((q_2, ..., q_n),E ∪ {q_1},T)
        else
            Result := ∅
            for all h :- b_1, ..., b_n ∈ BK do
                let ψ be a renaming substitution s.t var(b_i ψ) ∩ var(q_i) = ∅
                if ∃θ s.t. θ = mgu(q_1, hψ) then
                    Result := Result ∪
                        GenExplanationsR((b_1 ψθ, ..., b_n ψθ, q_2 θ, ..., q_n θ),E,T)
            return Result
```

probability of `calls(john)` *is not the sum of the probability of the explanations which is* 0.21*, but* 0.196*. The origin of the problem is that the explanations are not mutually exclusive, as both cover the possible world* $\{burglary, earthquake, al(john)\}$ *which has probability* 0.014 *which is therefore counted twice.*

The disjoint-sums-of-products problem arises because possible worlds might be a superset of multiple explanations. It is known that this problem, in general, is $\#P$-complete [Valiant, 1979]. Therefore, the explanations need to be made disjoint to calculate the probability. The straightforward solution is to use the principle of inclusion/exclusion, but this requires one to calculate an exponential number of probabilities. Therefore, we exploit the idea underlying *Shannon decomposition* to calculate the probability.

Assume that we want to calculate the probability of the formula, where the probability of x being true is $P(x)$. The Shannon decomposition is based on the observation that a propositional formula φ can be rewritten as disjunction of two co-factors $\varphi = (x \wedge \varphi[x \mapsto true]) \vee (\neg x \wedge \varphi[x \mapsto false])$, where $\varphi[x \mapsto v]$ arises from φ by replacing all occurrences of x by v. This is interesting, as the two co-factors are mutually exclusive and therefore

$$P(\varphi) = P(x)P(\varphi[x \mapsto true]) + (1 - P(x))P(\varphi[x \mapsto false]). \tag{3.3}$$

In terms of the set of explanations, this means that the explanations are split

into (1) those which require the ground fact x to be "false" and (2) those which do not require x to be "true". Algorithm 2 performs this calculation for a set of explanations. The run-time of this procedure depends largely on the order in which the literals x are chosen. Therefore, the explanations are first translated into a BDD. As outlined in Section 2.2 this BDD represents one Shannon decomposition of the explanations. Here the set E^+ corresponds to the positive child and E^- to the negative child of a node E. The reduction proces does not influence the probability. Merging isomorphic trees corresponds to tabling pre-computed results (cf. line 2 and 8). If nodes are removed, the probability of its two children are identical as they represent the same logical function. Therefore the probability of the removed node is marginalized out.

The reason to use BDDs is that existing implementations support a wide variety of heuristics, which ensures that in most cases the decomposition is small. Furthermore, due to reusing substructures, the algorithm can table results for sub-trees, resulting in a dynamic programming approach.

Algorithm 2 Calculate the probability of a set of explanations E

```
1:  function QueryProbability(Explanations E)
2:      if P(E) known then return P(E)
3:      if ∅ ∈ E then return 1
4:      if E = ∅ then return 0
5:      Select ground probabilistic fact p_i :: x_i
6:      E⁻ := {E_i\{not(x_i)} | E_i ∈ E and x_i ∉ E_i}        ▷ low-child in BDD
7:      E⁺ := {E_i\{x_i} | E_i ∈ E and not(x_i) ∉ E_i}        ▷ high-child in BDD
8:      store P(E) := p_i · QueryProbability(E⁺)
9:                    +(1 − p_i) · QueryProbability(E⁻)
10:     return P(E)
11:
```

3.3 CP-Logic

The second language we introduce is CP-Logic. While CP-Logic is strictly speaking not a probabilistic programming language in the sense of the distribution semantics, it can be easily mapped into such a language. This is done by representing selections, an important concept underlying the language, explicitly as probabilistic facts.

Compared to other probabilistic logic programming languages, CP-logic was developed as a knowledge representation framework [Vennekens et al., 2009]. CP-logic has a strong focus on causality and constructive processes: an interpretation is incrementally constructed by a process that adds facts to the interpretation which are probabilistic *effects* of other already given facts (the *causes*). More formally, a

model in CP-logic is defined as a set of (probabilistic) rules that represent causes and effects. A **CP-theory** is a set of rules. A rule r is of the form

$$(h_1 : p_1) \vee \ldots \vee (h_n : p_n) \longleftarrow b_1, \ldots, b_m \ ,$$

where the h_i are logical atoms, the b_i are atoms and $p_i \in [0, 1]$ probabilities such that $\sum_{i=1}^{n} p_i = 1$. As for definite clauses we will refer to $b_1, \ldots, b_m$ as the $body(r)$ of the rule and to $(h_1 : p_1) \vee \ldots \vee (h_n : p_n)$ as the $head(r)$ of the rule. Furthermore, we assume that the rules are **range-restricted**, that is, that all variables appearing in the head of the rule also appear in its body. The semantics of a CP-theory is given by the following probabilistic constructive process, which is closely related to the T_P operator (Section 2.3). Starting from the empty interpretation, at each step we consider all applicable rules, these are all groundings $r\theta$ of rules r such that $body(r\theta)$ holds in the current interpretation. For each of these groundings, one of the grounded head elements $h_1\theta, \ldots, h_n\theta$ of r is chosen randomly according to the distribution given by $p_1, \ldots, p_n$. The chosen head element is then added to the current interpretation, and the process is repeated until no more new atoms can be derived. Note that each grounding of a rule can only contribute a single head element.

Example 3.7 (Umbrella example in CP-Logic syntax) *The Example 3.2 presented in CP-Logic syntax. As CP-Logic does not support negation we need to explicitly represent the negation of rainy.*

The distributions for the first days are:

$$r0 = \quad rainy(0) : 0.2 \vee not\_rainy(0) : 0.8$$

$$w0 = \quad windy(0) : 0.4 \vee not\_windy(0) : 0.6$$

All subsequent days depend on the values sampled for the first day. To represent the argument for n-th day we need to encode the natural number n as $\underbrace{s(\ldots s(}_{n \text{ times}} 0) \ldots)$.

The rules are then

$$\begin{aligned}
n1 = &\quad rainy(s(X)) : 0.2 \vee not_rainy(s(X)) : 0.8 &&\longleftarrow not_rainy(X)\\
r1 = &\quad rainy(s(X)) : 0.8 \vee not_rainy(s(X)) : 0.2 &&\longleftarrow rainy(X)\\
w1 = &\quad windy(s(X)) : 0.4 \vee not_windy(s(X)) : 0.6 &&\longleftarrow windy(X)\\
n1 = &\quad windy(s(X)) : 0.4 \vee not_windy(s(X)) : 0.6 &&\longleftarrow not_windy(X) \ .
\end{aligned}$$

Finally the clause for taking an umbrella is

$$u = \quad umbrella(X) : 1 \longleftarrow rainy(X), not\_windy(X) \ .$$

To construct an interpretation CP-Logic starts from the empty interpretation $\{\}$ *and selects the applicable rules which are in this case* $r0$ *and* $w0$*. For example, after sampling* $rainy(0)$ *for* $r0$ *and* $not\_windy(0)$ *for* $w0$ *the rules applicable in the next iteration are* $r1$*,* $w1$*, and* u*. As this process would continue forever, CP-Logic theories are usually restricted to finite ground theories or theories with finite groundings.*

Before discussing the relationship of the ProbLog and CP-Logic we need to elaborate on the term "causal". The described constructive process provides a causal semantics for CP-Logic. The idea is similar to the semantic for *causal Bayesian networks* based on the notation of intervention [Pearl, 2009]. In a causal Bayesian network the parents of each node have to be its direct causes. The analysis of interventions can be redone in the context of CP-Logic [Vennekens et al., 2010]. Similarly to causal Bayesian networks in a CP-Theory the body of a clause should consists of the cause of the head. For example, even though the following two theories represent the same distribution they define different causal processes

$$0.5 :: rainy \qquad\qquad 0.5 :: wet$$

$$wet \longleftarrow rainy \qquad\qquad rainy \longleftarrow wet \ .$$

However, every CP-Logic theory defines a causal process.

While mapping a ProbLog program into CP-Logic is straightforward – definite clauses map directly, and probabilistic facts can be represented as rules with empty bodies – the inverse is slightly more involved. We first need to formalize the notation of a selection. A **selection** σ is a mapping from applicable ground rules $\mathbf{R}_t$ to head elements, associating each rule $r_i \in \mathbf{R}_t$ with one of its head elements $\sigma(r_i)$.

A selection σ allows one to map CP-Logic rules into probabilistic logic programs. For each CP-Logic rule $(h_1 : p_1) \vee \ldots \vee (h_n : p_n) \longleftarrow b_1, \ldots, b_m$, we introduce n clauses of the form $\mathtt{h_i} :- \mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{sel}(\mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{i})$. and one probabilistic fact $\mathtt{sel}(\mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{i})$. Let us assume that the atoms are ordered such that all atoms of one rule r occur consecutively, such that $A_{m+1} = sel(b_1, \ldots, b_m, 1), \ldots, A_{m+n} = sel(b_1, \ldots, b_m, n)$. Then

$$\begin{aligned} &P^{(m+n)}(A_1 = x_1, \ldots, A_{m+n} = x_{m+n}) \\ &= P^{(m+n)}(A_1 = x_1, \ldots, A_m = x_m) \\ &\quad \cdot P^r(A_{m+1} = x_{m+1}, \ldots, A_{m+n} = x_{m+n}) \end{aligned}$$

where

$$P^r(A_{m+1} = x_{m+1}, \ldots, A_{m+n} = x_{m+n}) = \begin{cases} 0 & \text{if } x_i = x_j = true \text{ and } i \neq j, \\ p_i & \text{else if } x_i = true \\ 0 & \text{otherwise} \end{cases}$$

All intermediate distributions and other orders can be recovered by marginalizing over all values of the atoms with the higher index.

Example 3.8 *Considering the clause*

$$r1[X/0] = rainy(s(0)) : 0.8 \vee not\_rainy(s(0)) : 0.2 \longleftarrow rainy(0) \ ,$$

the corresponding ground probabilistic logic program consists of

$$\mathcal{F} = \{\texttt{sel}(\texttt{rainy}(0), 1), \texttt{sel}(\texttt{rainy}(0), 2)\}$$

$$\mathcal{BK} = \{\texttt{rainy}(\texttt{s}(0)) :- \texttt{rainy}(0), \texttt{sel}(\texttt{rainy}(0), 1). \ ,$$

$$\texttt{not\_rainy}(\texttt{s}(0)) :- \texttt{rainy}(0), \texttt{sel}(\texttt{rainy}(0), 2).\}$$

and the distribution

$$P^{(2)}(\texttt{sel}(\texttt{rainy}(0), 1) = false, \texttt{sel}(\texttt{rainy}(0), 2) = false) = 0$$

$$P^{(2)}(\texttt{sel}(\texttt{rainy}(0), 1) = false, \texttt{sel}(\texttt{rainy}(0), 2) = true\ ) = 0.2$$

$$P^{(2)}(\texttt{sel}(\texttt{rainy}(0), 1) = true,\ \texttt{sel}(\texttt{rainy}(0), 2) = false) = 0.8$$

$$P^{(2)}(\texttt{sel}(\texttt{rainy}(0), 1) = true,\ \texttt{sel}(\texttt{rainy}(0), 2) = true)\ = 0$$

Stochastic Relational Processes

4

In this chapter, we study how to model stochastic relational processes, how to perform inference and how to learn such models. We introduce CPT-Logic for modelling such processes. We study inference and learning in two cases: first the fully observable case, and second the partially observable one, where we allow for hidden states.

One of the current challenges in artificial intelligence is the modeling of dynamic environments that change due to actions and activities people or other agents take, for example, modeling the action and activities of a cognitively impaired person [Pollack, 2005]. Such a model can be used to assist persons, using common patterns to generate reminders or detect potentially dangerous situations, and thus help improve living conditions.

As another example and one on which we shall focus in this part, consider a model of the environment in a *massively multiplayer online game* (MMOG). MMOGs are computer games that support thousands of players in complex, persistent, and dynamic virtual worlds. Such games form an ideal and realistic testbed for developing and evaluating artificial intelligence techniques, and are also interesting in their own right (cf. also [Laird and van Lent, 2000]). One challenge is, for

This chapter builds on [Thon et al., 2011], [Thon, 2009], [Thon et al., 2009] [Thon et al., 2008].

example, to build a dynamic probabilistic model of high-level player behavior, like players joining or leaving alliances and concerted actions by players within one alliance. Such a model of human cooperative behavior can be useful in several ways. Analysis of in-game social networks is not only interesting from a sociological point of view but could also be used to visualize aspects of the gaming environment or give advice to inexperienced players (e.g., which alliance to join). More ambitiously, the model could be used to build computer-controlled players that mimic the cooperative behavior of human players, to form alliances and jointly pursue goals, goals that would be impossible to attain otherwise. Mastering these social aspects of the game will be crucial to build smart and challenging computer-controlled opponents, which are currently lacking in most MMOGs. Finally, the model could also serve to detect non-human players in today's MMOGs — accounts which are played by automatic scripts to give one player an unfair advantage, which typically violate the rules of the game.

From a machine learning perspective, such a dynamic domain, has all the four main challenges: (chal1) a relatively large number of objects and relations is needed to build meaningful models, (chal2) the world state descriptions are inherently relational, as the interaction between (groups of) agents is of central interest, and the combined challenge (chal3, chal4) that the transition behavior of the world is strongly stochastic, as the defining element of environments such as MMOGs are interactions among *large* sets of agents. Thus, we need an approach that is both computationally efficient and able to represent complex relational state descriptions and stochastic world dynamics. In this setting, a relational state typically corresponds to a labeled (hyper)graph, and therefore the model can also be viewed as a stochastic model over sequences of graphs, cf. Figure 4.10.

Research in artificial intelligence has already contributed a rich variety of different modeling approaches, for instance, Markov models [Rabiner, 1989] and decision processes [Puterman, 1994], dynamic Bayesian networks [Ghahramani, 1998], statistical relational learning representations [Getoor and Taskar, 2007], STRIPS [Fikes and Nilsson, 1971] and probabilistic planning domain definition language (PPDDL) [Younes and Littman, 2004], noisy probabilistic relational planning rules [Zettlemoyer et al., 2005], RDDL [Sanner, 2010] etc. Most of the existing approaches that support reasoning about uncertainty that is, satisfy requirement (chal3)) employ essentially propositional representations (for instance, dynamic Bayesian networks, Markov models). Therefore, they are not able to represent complex relational worlds, and do not satisfy requirement (chal2). A class of models that integrates logical or relational representations with methods for reasoning about uncertainty is considered within statistical relational learning [Getoor and Taskar, 2007] and probabilistic inductive logic programming [De Raedt et al., 2008] (for instance, Markov Logic [Richardson and Domingos, 2006], CP-logic [Vennekens et al., 2006], or Bayesian Logic Programs [Kersting and De Raedt, 2007]). However, inference and learning often cause significant computational

problems in realistic applications, and hence, such methods do not satisfy requirement (chal1). Languages developed to solve planning problems like STRIPS, PPDDL, noisy probabilistic relational planning rules satisfy (chal2),(chal3) and (chal4), but they typically model only a single probabilistic process. The recently introduced formalism RDDL overcomes this shortcoming. Is well suited to model the domains introduced in this chapter, but still requires that the number of objects in the world does not change. The relationship to PPDDL will be discussed further in Appendix 8.1.

We want to alleviate this situation, by contributing a novel representation, called CPT-L (for **C**ausal **P**robabilistic **T**ime-**L**ogic), that occupies an intermediate position in this expressiveness/efficiency trade-off. A CPT-L model essentially defines a probability distribution over sequences of Herbrand interpretations. Herbrand interpretations are relational state descriptions that are typically used in planning and many other applications of artificial intelligence. CPT-L can be considered a variation of CP-logic [Vennekens et al., 2006], an expressive logic for modeling causality. It does this by focusing on the sequential aspect and deliberately avoiding the complications that arise when dealing with hidden states. Thus CPT-L is more restricted, but also more efficient to use than alternative formalisms within the artificial intelligence and statistical relational learning literature. While we do not allow hidden states the model still contains hidden random variables. The dependencies of these hidden random variables are limited to random variables in the same transition. This trades efficiency for the expressivity required to be able to model interesting problems.

Limiting the language by not allowing hidden variables, allows for efficient exact inference and learning algorithms. However, this limitation sometimes prevents us from solving interesting problems. Therefore, we also investigate, how a CPT-L model can be used in the presence of hidden variables. An approximate inference algorithm for the filtering problem will be given as well.

This part is organized as follows: Section 4.1 introduces the CPT-L framework and several extensions, such as higher order dependencies, representing hidden variables, and the representation of PPDDL domains. Section 4.2 addresses inference and parameter estimation; and Section 4.3 presents the results of experiments in several (artificial and real-world) domains. Finally, we discuss related work in Section 4.4, before concluding and touching upon future work in Section 4.5. Proofs of the main theorems are contained in Appendix 8.2 and 8.3.

4.1 Representation

The requirements (chal1) and (chal2) formulated in the previous section imply that the system must be able to represent world states that are relational and consist of

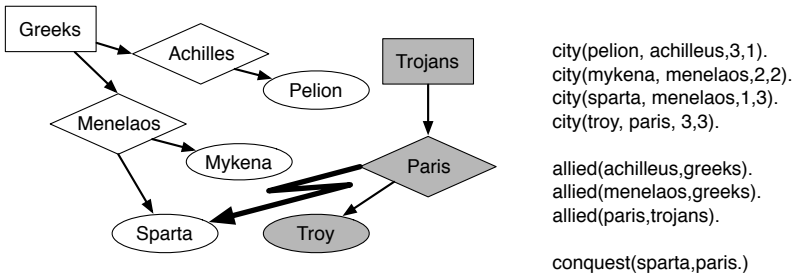


Figure 4.1: Example for the state of a multiplayer game represented as a graph structure and, equivalently, as a logical interpretation. The rectangles in graphical representation refer to alliances, diamonds to players, and ellipsis to cities. The last two arguments of city in the logical representation refer to the location of the city.

a large number of objects. Such complex world states can be described in terms of *interpretations*. An interpretation I is a set of ground facts $\{a_1, \ldots, a_N\}$. These ground facts can represent objects in the current world state, their properties, and any relationship between objects. As an example, reconsider the representation of the state of a multiplayer game in terms of an interpretation as depicted in Figure 4.1 and the example of the blocksworld in Figure 4.2.

The semantics of our framework is based on CP-logic (Section 3.3). The strong focus on causality and constructive processes of CP-Logic makes it well suited to express stochastic processes. This constructive process is as follows: an interpretation is incrementally constructed by a process that adds facts to the interpretation which are probabilistic *outcomes* of other already given facts (the *causes*).

CPT-L combines the semantics of CP-logic with that of (first-order) Markov processes. This corresponds to the assumption that for any sequence of interpretations there is an underlying generative process that constructs the next interpretation from the current one. A (discrete-time) *stochastic process* defines a

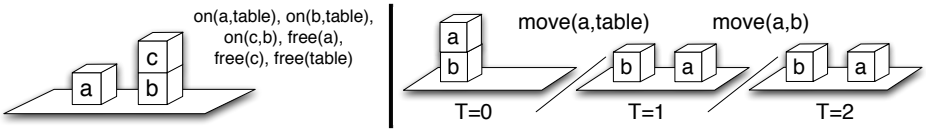


Figure 4.2: Example of a state in blocksworld represented as an image and as a logical interpretation (left), and a sequence of states in the blocksworld domain (right).

distribution $P(X_1, \ldots, X_T)$ over a sequence of random variables $X_1, \ldots, X_T$ that characterize the state of the world at time $t = 1, \ldots, T$. We are interested in the case where X is a relational state description, that is, a **relational stochastic processes**. A relational stochastic process defines a distribution $P(I_0, \ldots, I_T)$ over sequences of interpretations $I_0, \ldots, I_T$, where interpretation I_t describes the state of the world at time t. Thus, the random variable X_t describing the state of the process at time t is an interpretation, that is, a structured state.

The main idea behind CPT-L is to apply the causal probabilistic framework of CP-logic to stationary Markov processes, by assuming that the state of the world at time $t + 1$ is a probabilistic outcome of the state of the world at time t. The constructive probabilistic process is thus unfolded over time, such that observed facts in interpretation I_t (probabilistically) cause other facts to be observed in I_{t+1}. In this setting, the first-order Markov assumption states that causal influences only stretch from I_t to I_{t+1}, but not further into the future. More formally, we define a **CPT-theory** as follows:

Definition 4.1 *A* **CPT-theory** *is a set of rules of the form*

$$(h_{1,1} \wedge \ldots \wedge h_{1,k_1} : p_1) \vee \ldots \vee (h_{n,1} \wedge \ldots \wedge h_{1,k_n} : p_n) \longleftarrow b_1, \ldots, b_m$$

where the $h_{i,j}$ are logical atoms, $p_i \in [0, 1]$ are probabilities s.t. $\sum_{i=1}^{n} p_i = 1$, and the b_l are literals (i.e., atoms or their negation). For a rule r, $head(r)$ is $(h_{1,1} \wedge \ldots \wedge h_{1,k_1} : p_1) \vee \ldots \vee (h_{n,1} \wedge \ldots \wedge h_{1,k_n} : p_n)$ and $body(r) = b_1, \ldots, b_m$.

When referring to the semantic of a specific CPT-theory, we usually use the term **CPT-L model**. A conjunction $h_{i,1} \wedge \ldots \wedge h_{i,k_i}$ in $head(r)$ will also be called a *head element*, and its probability p_i will be denoted by $P(h_{i,1} \wedge \ldots \wedge h_{i,k_i} \mid r)$. The meaning of a rule is that whenever $b_1\theta, \ldots, b_m\theta$ holds for a substitution θ in the current state I_t, exactly one head element $h_{i,1}\theta \wedge \ldots \wedge h_{i,k_i}\theta$ is chosen from $head(r)$ and all its atoms $h_{i,j}\theta$ are added to the next state I_{t+1}.

Note that in contrast to CP-logic, outcomes in CPT-L can be conjunctions of facts rather than individual facts. This is needed to represent causes with multiple outcomes in the next time step. In CP-logic, such multiple outcomes can be easily simulated using a set of rules of the form $h_{i,j} : 1 \longleftarrow h_i$ for $j = 1, \ldots, k_i$ that expand a single head element h_i into a conjunction $h_{i,1}, \ldots, h_{i,k_i}$. However, in CPT-L no new facts can be derived within one state I_t, thus such an expansion is not possible and conjunctions are needed to represent multiple outcomes.

Example 4.1 *Consider the following CPT-theory for the* blocks world *domain [Russell and Norvig, 2003]:*

$$r_1 = free(X) : 1.0 \longleftarrow free(X), \neg move(Y, X)$$

$$r_2 = on(X, Y) : 1.0 \longleftarrow on(X, Y), \neg move(X, Z), free(Z)$$

$$r_3 = (on(A, B) \wedge free(C) : 0.9) \vee (on(A, C) \wedge free(B) : 0.1) \longleftarrow$$
$$free(A), free(B), on(A, C), move(A, B).$$

The first two rules represent frame axioms, namely that a block stays free if no other block is moved upon it, and that blocks stay on each other unless they are moved. The third rule states that if we try to move block A on block C this succeeds with a probability of 0.9*.*

We now show how a CPT-theory defines a distribution over sequences $I_0, \ldots, I_T$ of relational interpretations. Let us first define the concept of an **applicable** rule r in an interpretation I_t. Consider a CPT rule $c_1 : p_1 \vee \ldots \vee c_n : p_n \longleftarrow b_1, \ldots, b_m$. Let θ denote a substitution that grounds the rule r, and let $r\theta$ denote the grounded rule. A rule r is applicable in I_t if and only if there exists a substitution θ such that $body(r)\theta = b_1\theta, \ldots, b_m\theta$ is true in I_t, denoted $I_t \models b_1\theta, \ldots, b_m\theta$. We will most often talk about ground rules that are applicable in an interpretation.

One of the main features of CPT-theories is that they are easily extended to include *background knowledge*. The background knowledge B can be any logic program (cf. [Bratko, 1990]). When working with background knowledge, the state I_t is represented by a set of facts and a ground rule is applicable in a state I_t if $b_1\theta, \ldots, b_m\theta$ can be inferred from I_t together with the background knowledge B. More formally, a ground rule is applicable if and only if $I_t \cup B \models b_1\theta, \ldots, b_m\theta$. To simplify the notation during the elaboration of our probabilistic semantics we largely ignore the use of background knowledge.

Given a CPT-Theory $\mathcal{T}$, the set of all applicable ground rules in state I_t will be denoted as $\mathbf{R}_t$. That is, $\mathbf{R}_t = \{r\theta \mid r \in \mathcal{T}, r\theta \text{ applicable in } I_t\}$. Each ground rule applicable in I_t will cause one of its grounded head elements to be selected, and the resulting atoms to become true in I_{t+1}. More formally, let $\mathbf{R}_t = \{r_1, \ldots, r_k\}$ with the ground rules r_i. A **selection** σ is a mapping from applicable ground rules in $\mathbf{R}_t$ to head elements, associating each rule $r_i \in \mathbf{R}_t$ with one of its head elements $\sigma(r_i)$. Note that $\sigma(r_i)$ is a conjunction of ground atoms. Each selection corresponds to a random outcome of a random variable associated with a ground rule. The probability of σ is simply the product of the probabilities of selecting the respective head elements, that is,

$$P(\sigma) = \prod_{i=1}^{k} P(\sigma(r_i) \mid r_i) \ , \tag{4.1}$$

where $P(\sigma(r_i) \mid r_i)$ is the probability associated with head element $\sigma(r_i)$ in the rule r_i.

A selection σ defines which head element is selected for every rule, and thus determines a successor interpretation I_{t+1}, that simply consists of all atoms appearing in selected head elements. More formally,

$$I_{t+1} = \bigwedge_{i=1}^{k} \sigma(r_i) \ ,$$

where, abusing notation, we have denoted an interpretation as a conjunction of atoms rather than a set of atoms. We shall say that σ *yields* I_{t+1} from I_t, denoted $I_t \stackrel{\sigma}{\rightarrow} I_{t+1}$, and define

$$P(I_{t+1}|I_t) = \sum_{\sigma : I_t \stackrel{\sigma}{\rightarrow} I_{t+1}} P(\sigma) \ . \tag{4.2}$$

That is, the probability of a successor interpretation I_{t+1} given an interpretation I_t is computed by summing the probabilities of all selections yielding I_{t+1} from I_t. Note that $P(I_{t+1} \mid I_t) = 0$ if no selection yields I_{t+1}.

Example 4.2 *Consider the theory*

$$\begin{array}{lrl} r_1 & = & p(X) : 0.2 \vee q(X) : 0.8 \quad \longleftarrow q(X) \\ r_2 & = & p(a) : 0.5 \vee (q(b) \wedge q(c)) : 0.5 \quad \longleftarrow \neg q(b) \\ r_3 & = & p(X) : 0.7 \vee nil : 0.3 \quad \longleftarrow p(X) \end{array}$$

Starting from $I_t = \{p(a)\}$ *only the rules* r_2 *and* r_3 *are applicable, so* $\mathbf{R}_t = \{r_2, r_3\theta\}$, *with* $\theta = [X/a]$. *The set of possible selections is* $\Gamma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ *with*

$$\begin{array}{ll} \sigma_1 = \{(r_2, p(a)), (r_3\theta, p(a))\} & \sigma_2 = \{(r_2, q(b) \wedge q(c)), (r_3\theta, p(a))\} \\ \sigma_3 = \{(r_2, p(a)), (r_3\theta, nil)\} & \sigma_4 = \{(r_2, q(b) \wedge q(c)), (r_3\theta, nil)\} \end{array}$$

The possible successor states I_{t+1} *are therefore*

$$I^1_{t+1} = \{p(a)\} \text{ with } P(I^1_{t+1} \mid I_t) = P(\sigma_1) + P(\sigma_3) = 0.5 \cdot 0.7 + 0.5 \cdot 0.3 = 0.5$$

$$I^2_{t+1} = \{q(b), q(c)\} \text{ with } P(I^2_{t+1} \mid I_t) = P(\sigma_4) = 0.5 \cdot 0.3 = 0.15$$

$$I^3_{t+1} = \{p(a), q(b), q(c)\} \text{ with } P(I^3_{t+1} \mid I_t) = P(\sigma_2) = 0.5 \cdot 0.7 = 0.35$$

As for propositional Markov processes, the probability of a sequence $I_0, \ldots, I_T$ given an initial state I_0 is defined by

$$P(I_0, \ldots, I_T) = P(I_0) \prod_{t=0}^{T} P(I_{t+1} \mid I_t). \tag{4.3}$$

Intuitively, it is clear that this defines a distribution over all sequences of interpretations of length T as in the propositional case. More formally, inductive application of the product rule yields the following theorem:

Theorem 4.1 (Semantics of a CPT-theory) *A CPT-theory $\mathcal{T}$ defines a discrete-time stochastic process, Given an initial state,, a and therefore for $T \in \mathbb{N}$ a distribution $P_{\mathcal{T}}(I_0, \ldots, I_T)$ over sequences of interpretations of length T.*

4.1.1 Relaxing the Markov Assumption

The CPT-L model described so far is based on a first-order Markov assumption Equation (8.2). As for propositional Markov processes, it is straightforward to relax this assumption and allow higher-order dependencies such that

$$P(I_0, \ldots, I_T) = P(I_0) \prod_{t=0}^{T} P(I_{t+1} \mid I_{t-n+1}, \ldots, I_t) \ ,$$

where $n > 1$ is the model order. In particular, for $n = \infty$, we have a full-history model given by

$$P(I_0, \ldots, I_T) = P(I_0) \prod_{t=0}^{T} P(I_{t+1} \mid I_0, \ldots, I_t). \tag{4.4}$$

For propositional Markov processes, a naïve representations of the distribution $P(I_{t+1} \mid I_{t-n+1}, \ldots, I_t)$ leads to a number of model parameters that is exponential in n. Thus, higher-order models typically require additional assumptions (as in Mixed Memory Markov Models [Saul and Jordan, 1999]) and/or regularization to avoid overfitting during learning and excessive computational complexity. However, in CPT-L we can easily take into account all previous interpretations when constructing a successor interpretation without a combinatorial explosion in model complexity. The idea is to extend rule conditions to match on all previous interpretations. This can be realized by aggregating all previous interpretations $I_t, I_{t-1}, \ldots, I_0$ using fluents (facts extended with an additional argument for the timepoint), and then matching on the aggregated history. An example of a fluent is $\texttt{weather}(\texttt{rainy}, \texttt{s}(\texttt{0}))$, where $\texttt{s}(\texttt{0})$ is the timepoint, and $\texttt{weather}(\texttt{rainy}, \texttt{X})$ aggregates the past days. More formally, let $\mathcal{F}(I, t)$ denote the interpretation I where all facts have been extended by an additional argument t, as in $\mathcal{F}(I, 0) = \{p(0, a), q(0, b)\}$ for $I = \{p(a), q(b)\}$. Now define the aggregated history as

$$I_{[0,t]} = \bigcup_{t'=0}^{t} \mathcal{F}(I_{t'}, t' - t).$$

CPT-L rules are still of the form

$$r = c_1 : p_1 \vee \ldots \vee c_n : p_n \longleftarrow b_1, \ldots, b_m$$

where the head elements c_i are conjunctions and the p_i probabilities as in Definition 4.1, but body literals b_i now match on the interpretation $I_{[0,t]}$. According to Equation (4.4), we now need to construct a successor interpretation I_{t+1} given a history of interpretations $I_t, I_{t-1}, \ldots, I_0$, or, equivalently, giving the aggregated history $I_{[0,t]}$. In this new setting, a rule r is **applicable** given $I_t, I_{t-1}, \ldots, I_0$ if and only if there is a grounding θ such that $I_{[0,t]} \models b_1\theta, \ldots, b_m\theta$. As before, we probabilistically select for every applicable rule a grounded head element $c_i\theta$ and add its atoms to I_{t+1}.

Example 4.3 *Reconsider Example 4.2. In the new setting, rules r_1, r_2, r_3 can be written as*

$$\begin{aligned} r_1 &= p(X) : 0.2 \vee q(X) : 0.8 \longleftarrow q(0, X) \\ r_2 &= p(a) : 0.5 \vee (q(b) \wedge q(c)) : 0.5 \longleftarrow \neg q(0, b) \\ r_3 &= p(X) : 0.7 \vee nil : 0.3 \longleftarrow p(0, X) \end{aligned}$$

Assume we are given a history $I_1 = \{p(a)\}, I_0 = \{q(b), q(c)\}$ and need to compute $P(I_2 \mid I_1, I_0)$. The joint interpretation is

$$I_{[0,1]} = \{p(0, a), q(-1, b), q(-1, c)\},$$

where $p(0, a)$ is from interpretation I_1 and the other two are from I_0, where $t' - t = -1$. The possible successor interpretations I_2 are, of course, the same as in Example 4.2. Rule r_1 could be changed to

$$r_1 = p(X) : 0.2 \vee q(X) : 0.8 \longleftarrow q(T, X)$$

to make it applicable whenever $\{q(X)\}$ succeeds in any earlier interpretation (not necessarily the previous one).

As the first-order Markov variant of CPT-L discussed in Section 4.1 is a special case of the more general variant discussed in this section, we will for the rest of this thesis only consider full-history models. The conditional successor distribution $P(I_{t+1} \mid I_t, \ldots, I_0)$ will also be denoted by $P(I_{t+1} \mid I_{[0,t]})$.

4.1.2 Representation of Unobserved Facts

Up to now, we assumed that there are no unobserved facts, therefore the only hidden variables are the selections of the rules. The assumption that all variables are observable is not uncommon as it allows efficient inference (n-grams or naïve Bayes), which often outweighs the imprecision. On the other hand some tasks require unobserved facts.

Reconsider the example of recognizing activities of cognitively impaired persons, where the goal is to assist by generating reminders and detecting potentially

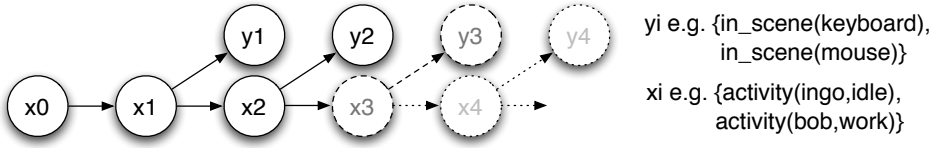


Figure 4.3: Graphical representation of an HMM. States and observations are in our case Herbrand interpretations.

dangerous situations. To be able to do so the system has to infer the intention or the activities of a person from features derived from sensory information. The typical model used in such processes are Hidden Markov Models (HMM) [Rabiner, 1989] and their generalizations like factorial HMMs [Ghahramani and Jordan, 1997], or Dynamic Bayesian Networks (DBN) [Ghahramani, 1998]. These models can represent the intentions and/or activities by using a hidden state. However none of these models can represent relational data. Given that relational representations are not only useful but also required for many applications, we introduce Hidden CPT-L (HCPT-L). HCPT-L allows to define relational models in the presence of unobserved facts. Analogous to the idea underlying CPT-L which is basically a Markov chain where states are Herbrand interpretations and transition are defined in terms of probabilistic logic, a HCPT-L model is basically a HMM where states and observations are Herbrand interpretations and transition- and observation-probabilities are defined in terms of a probabilistic logic (cf. Fig 4.3). More formally:

Definition 4.2 *A* **HCPT-L model** *consists like a CPT-L model of a set of rules of the form*

$$r = (h_1 : p_1) \vee \ldots \vee (h_n : p_n) \leftarrow b_1, \ldots, b_m$$

where the $p_i \in [0, 1]$ *form a probability distribution such that* $\sum_i p_i = 1$*,* h_i *are logical atoms,* b_i *are literals (i.e. atoms or their negation). The key difference is that the rules are grouped into the transition model and the observation model. The head atoms of the observation model and the body atoms of all rules need to be mutually exclusive.*

For convenience we indicate the observation model by defining atoms as observable.

Example 4.4 *Consider the following example rules that models the current activity:*

$$\begin{array}{rcccl} r & = & a(P,X) : 0.8 \vee a(P,drink) : 0.1 \vee a(P,work) : 0.1 & \leftarrow & a(P,X). \\ od & = & ois(can) : 0.7 \vee nil : 0.3 & \leftarrow & a(P,drink). \\ ow & = & ois(pen) : 0.7 \vee nil : 0.3 & \leftarrow & a(P,work). \\ & & \textit{observable } ois/1 & & \end{array}$$

The first rule states that person P will continue its current activity X with probability 0.8 *or switch to one of the activities work or drink with probability* 0.1*. The second and third rule specifies: if someone works/drinks one can observe a pen/can. The unary predicate $ois/1$ and therefore rule od and ow are declared to be observable.*

The predicate $a/2$ would typically be unobservable, whereas predicates like $pose/2$, $movement/2$, $object\_in\_scene/1$ ($ois/1$) would be observable. A rule is called un-/observable according to the observability of the predicates in the head.

In the remainder I_k denotes the set of all unobservable facts, as these are the states of the Markov chain. The observations which consists of the observable facts true at time point k are denoted by O_k. The probability of a hidden state sequence together with a sequence of observations follows directly from the semantics of CPT-L. From the viewpoint of CPT-L the observation O_k of time-point k belongs to the successor state as both are caused by the current state. Consequently we will use $P_o(O_{t+1}|I_t)$ as distribution over the observable predicates and $P_s(I_{t+1}|I_t)$ as distribution over the unobserved predicates.

To continue our example assume that the current state is $I_i = \{a(ann, work), a(bob, work)\}$ then the applicable rules are $r[P/ann, X/work]$, $r[P/bob, X/work]$, $ow[P/ann]$, $od[X/bob]$. For each rule a head element is selected and either added to the successor state or the observation if it is defined to be observable. The next state is $I_{i+1} = \{a(ann, work), a(bob, drink)\}$ and the observation is $O_i = \{ois(pen)\}$ for example with probability $(0.8 + 0.1) \cdot 0.1 \cdot (0.7 + (0.3 \cdot 0.7))$.

4.2 Inference And Parameter Estimation in CPT-L And HCPT-L

In the previous section we have introduced the CPT-L language. This language formalizes models of stochastic relational processes. To be able to apply these models in practice we will introduce algorithms for solving the standard inference tasks. As for other probabilistic models, we can now formulate several computational tasks for the introduced CPT-L model:

- **Sampling** (Section 4.2.1): sample sequences of interpretations $I_1, \ldots, I_T$ from a given CPT-theory $\mathcal{T}$ and initial interpretation I_0.
- **Inference** (Section 4.2.2): given a CPT-theory $\mathcal{T}$ and a sequence of interpretations $I_0, \ldots, I_T$, compute $P(I_0, \ldots, I_T \mid \mathcal{T})$.
- **Parameter Estimation** (Section 4.2.4): given the structure of a CPT-theory $\mathcal{T}$ and a set D of sequences of interpretations, compute the

maximum-likelihood parameters $\pi^* = \arg\max_\pi P(D \mid \mathcal{T}(\pi))$, where π are the parameters of $\mathcal{T}$.

- **Prediction** (Section 4.2.5): Let $\mathcal{T}$ be a CPT-theory, $I_0, \ldots, I_t$ a sequence of interpretations, and F a first-order query that represents a certain property of interest. Compute the probability that F holds at time $t + d$, that is, $P(I_{t+d} \models F \mid \mathcal{T}, I_0, \ldots, I_t)$.
- **Filtering** (Section 4.2.6): Let $\mathcal{T}$ be a HCPT-theory, $I_0, \ldots, I_t$, $O_0, \ldots, O_{t+k}$ sequences of interpretations representing starting state(s) and observations, and F a first-order query that represents a certain property of interest. Compute the probability that F holds at time $t + k$, that is, $P(I_{t+k} \models F \mid \mathcal{T}, I_0, \ldots, I_t, O_0, \ldots, O_{t+k})$.

Algorithmic solutions for solving these tasks will now be presented.

4.2.1 Sampling

The first inference task we solve is sampling, there the goal is to sample sequences of interpretations $I_1, \ldots, I_T$ from a given CPT-theory $\mathcal{T}$ and initial interpretation I_0. This is interesting for two reasons. First of all it allows one to inspect what a sequence specified by a model looks like. Second, as we will see later, sampling can also be utilized to predict future states. Sampling from a CPT-theory is straightforward due to the causal semantics employed in the underlying CP-logic framework. Let $\mathcal{T}$ be a CPT-theory, and let I_0 be an initial interpretation. According to Equation (4.4), we can sample from the joint distribution $P(I_1, \ldots, I_T \mid I_0)$ by successively sampling I_{t+1} from the distribution $P(I_{t+1} \mid I_{[0,t]})$ for $t = 0, \ldots, T-1$. This can be done directly using the constructive process that defines the semantics of CPT-L. We start with the empty interpretation $I_{t+1} = \{\}$, and first find all groundings $r\theta$ of rules $r \in \mathcal{T}$ that are applicable in $I_{[0,t]}$. For each ground rule $r\theta$, we then randomly select one of its head elements $c \in \mathit{head}(r\theta)$ according to the probability distribution over head elements for that rule. The head element c is a conjunction of atoms, which need to be added to I_{t+1}. After adding one such conjunction for each applicable rule, we have randomly sampled I_{t+1} from the desired distribution.

4.2.2 Inference for CPT-Theories

The second inference problem we study is that of computing the probability of a sequence given a model. More formally, let $\mathcal{T}$ be a given CPT-theory, and $I_0, \ldots, I_T$ be a sequence of interpretations. Utilizing Equation (4.4), the crucial task for solving the inference problem is to compute $P(I_{t+1} \mid I_{[0,t]})$ for $t = 0, \ldots, T-1$.

According to Equation (8.1), this involves marginalizing over all selections yielding I_{t+1} from $I_{[0,t]}$. However, the number of possible selections σ can be exponential in the number of ground rules $|\mathbf{R}_t|$ applicable in $I_{[0,t]}$, so a naïve generate-and-test approach is infeasible. Instead, we present an efficient approach for computing $P(I_{t+1} \mid I_{[0,t]})$ without explicitly enumerating all selections yielding I_{t+1}, this is strongly related to the inference technique discussed for Problog [De Raedt et al., 2007]. The problem of computing $P(I_{t+1} \mid I_{[0,t]})$ is first translated into a CNF formula over Boolean variables such that satisfying assignments correspond to selections yielding I_{t+1}. The formula is then compactly represented as a binary decision diagram (BDD), and $P(I_{t+1} \mid I_{[0,t]})$ efficiently computed from the BDD using dynamic programming. Although finding satisfying assignments for CNF formulae is a hard problem in general, the key advantage of this approach is that existing, highly optimized BDD software packages with advanced heuristics can be used.

Conversion to CNF

The conversion of an inference problem $P(I_{t+1} \mid I_{[0,t]})$ to a CNF formula f is done as follows:

1. Initialize $f := true$
2. Let $\mathbf{R}_t$ denote the set of applicable ground rules in $I_{[0,t]}$. Rules $r \in \mathbf{R}_t$ are of the form $r = c_1 : p_1, \ldots, c_n : p_n \longleftarrow b_1, \ldots, b_m$, where c_i are conjunctions of literals (see Definition 4.1).
3. For all rules $r \in \mathbf{R}_t$ do:
   (a) $f := f \wedge (r.c_1 \vee \ldots \vee r.c_n)$, where $r.c_i$ denotes a new (propositional) Boolean variable whose unique name is the concatenation of the name of the rule r with the head element c_i.
   (b) $f := f \wedge (\neg r.c_i \vee \neg r.c_j)$ for all $i \neq j$
4. For all facts $l \in I_{t+1}$
   (a) Initialize $g := false$
   (b) for all $r \in \mathbf{R}_t$ and $c_i : p_i \in head(r)$ such that l is one of the atoms in the conjunction c_i do $g := g \vee r.c_i$
   (c) $f := f \wedge g$
5. For all variables $r.c$ appearing in f such that one of the atoms in the conjunction c is not true in I_{t+1} do $f = f \wedge \neg r.c$

A Boolean variable $r.c$ in f represents that head element c was selected in rule r. A selection σ thus corresponds to an assignment of truth values to the variables $r.c$, in which exactly one $r.c$ is true for every rule r. The construction of f ensures that all satisfying assignments for the formula f correspond to selections yielding I_{t+1}, and vice versa. Specifically, Step 3 of the algorithm assures that selections are obtained (that is, exactly one head element is selected per rule), Step 4 assures that the selection generates the interpretation I_{t+1}, and Step 5 assures that no facts are generated that do not appear in I_{t+1}. Thus, we have a one-to-one correspondence between satisfying assignments for the formula f and selections yielding I_{t+1}.

Example 4.5 *The following formula f is obtained for the CPT-theory given in Example 4.2*

$$\begin{array}{rcrl} r_1 & = & p(X):0.2 \vee q(X):0.8 & \longleftarrow q(X) \\ r_2 & = & p(a):0.5 \vee (q(b) \wedge q(c)):0.5 & \longleftarrow \neg q(b) \\ r_3 & = & p(X):0.7 \vee \mathit{nil}:0.3 & \longleftarrow p(X) \end{array}$$

and the transition $\{p(a)\} \to \{p(a)\}$:

$$\underbrace{(r2.c_{21} \vee r2.c_{22}) \wedge (r3.c_{31} \vee r3.c_{32})}_{3.a}$$

$$\wedge \underbrace{(\neg r2.c_{21} \vee \neg r2.c_{22})) \wedge (\neg r3.c_{31} \vee \neg r3.c_{32})}_{3.b}$$

$$\wedge \underbrace{(r2.c_{21} \vee r3.c_{31})}_{4} \wedge \underbrace{\neg r2.c_{22}}_{5}$$

where $c_{21} = p(a)$, $c_{22} = q(b) \wedge q(c)$, $c_{31} = p(a)$ and $c_{32} = \mathit{nil}$ are the head elements of rules r_2 and r_3. The parts of the formula are annotated with the steps in the construction algorithm that generated them.

The size of the CNF and, therefore, the complexity of this procedure is linear in $\mathbf{R}_t$, I_t and I_{t+1}. Furthermore, it is exponential in the maximal number of head elements, but this factor is typically dominated by one of the other ones.

Calculate Probability By Means of BDDs

From the formula f, a *(reduced ordered) binary decision diagram* (Section 2.2) is constructed. Figure 4.4 shows the BDD resulting from the formula f given in Example 4.5.

From the BDD graph, $P(I_{t+1} \mid I_{[0,t]})$ can be computed in linear time using dynamic programming. The resulting algorithm is strongly related to the algorithm for

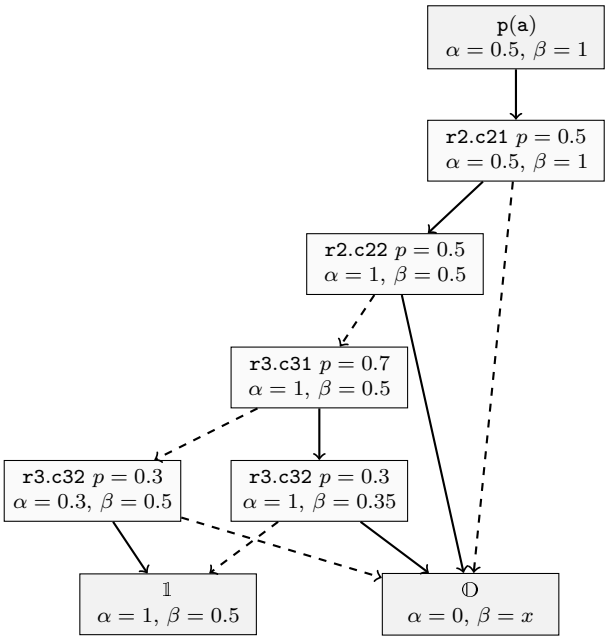


Figure 4.4: A BDD representing the formula f given in Example 4.5. The root node indicates the observed interpretation. The terminal nodes represent whether the path starting at the root node yields this interpretation. The other nodes are annotated with the rule r and head element c they represent, indicating the Boolean variable $r.c$ used in f. If a node is exited by a solid edge, the corresponding variable is assigned the value true, otherwise it is assigned the value false. Also given are upward probabilities $\alpha(N)$ and downward probabilities $\beta(N)$ for all nodes N, as $(\alpha(N) \mid \beta(N))$.

inference in ProbLog theories (Section 3.2). The main difference is that in ProbLog a propositional variable represents both the positive and negative outcome, whereas for CPT-L the negative outcome is explicitly represented by another variable. First note that there is a one-to-one correspondence between paths in the BDD from the root to the 1-terminal and selections yielding I_{t+1}, where the path indicates which of the Boolean variables $r.c$ in f are assigned the value true, or equivalently, which head element c has been selected for rule r. To see this, consider Step 3 of the algorithm for converting a given inference problem into the BDD. It ensures that exactly one head element is chosen for every rule. Thus, in the BDD representation, every path to the 1-terminal must pass through all Boolean variables; otherwise, the state of one variable could be altered, violating the constraint encoded in Step 3 of the conversion algorithm.

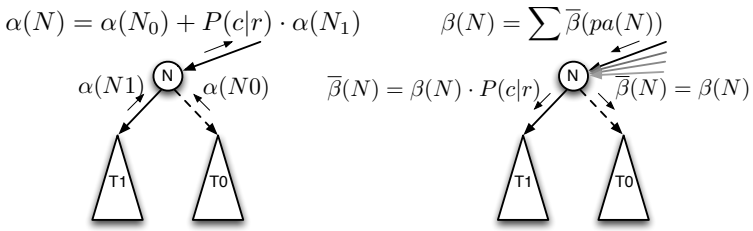


Figure 4.5: Calculation of upward and downward probabilities for internal nodes in the BDD.

We now recursively define for every node N in the BDD an **upward probability** $\alpha(N)$ as follows:

1. The upward probabilities for terminal nodes are defined as

   $$\alpha(\text{0-terminal}) = 0 \text{ and } \alpha(\text{1-terminal}) = 1.$$

2. Let N be a node in the BDD representing the Boolean variable $r.c$, with r a rule and c one of its head elements. Let N^-, N^+ denote the children of N, with N^- on the negative and N^+ on the positive branch. Then

   $$\alpha(N) = \alpha(N^-) + P(c \mid r)\alpha(N^+).$$

Furthermore, we recursively define a **downward probability** $\beta(N)$ as follows:

1. The downward probability of the root node is defined as

   $$\beta(\text{root}) = 1. \tag{4.5}$$

2. Let N be a non-root node in the BDD. Let $N_1, \ldots, N_k$ denote the parents of N, with $N_1, \ldots, N_l = pa^+(N)$ reaching N by their positive branch and $N_{l+1}, \ldots, N_k = pa^-(N)$ reaching N by their negative branch. Then

   $$\beta(N) = \sum_{i=1}^{l} \beta(N_i)P(c_i \mid r_i) + \sum_{i=l+1}^{k} \beta(N_i) \tag{4.6}$$

   where $r_i.c_i$ is the Boolean variable associated with node N_i.

The definition of upward and downward probabilities is visualized in Figure 4.5. The values $\alpha(N)$ and $\beta(N)$ can be interpreted as probabilities of partial selections, which are determined by the path from the 1-terminal (α) or the root (β) to the

node N. They roughly correspond to the forward-backward probabilities used for inference in hidden Markov models [Rabiner, 1989], or inside-outside probabilities used in stochastic context free grammars.

The following theorem states that the desired probability $P(I_{t+1} \mid I_{[0,t]})$ for inference can be easily obtained given the upward and downward probabilities:

Theorem 4.2 *Let $\mathcal{B}$ be a BDD resulting from the conversion of an inference problem $P(I_{t+1} \mid I_{[0,t]})$, annotated with upward and downward probabilities as defined above, and let*

$$\Gamma = \{\sigma \mid I_{[0,t]} \xrightarrow{\sigma} I_{t+1}\}$$

be the set of selections yielding I_{t+1}. Then

$$\begin{aligned} \alpha(root) &= \sum_{\sigma \in \Gamma} P(\sigma) \\ &= P(I_{t+1} \mid I_{[0,t]}). \end{aligned} \tag{4.7}$$

A proof of the theorem is given in Appendix 8.3. Note that the downward probabilities will only be needed for the parameter estimation algorithm discussed in Section 4.2.4. Computing upward and downward probabilities from their recursive definitions is straightforward, thus Theorem 4.2 concludes the description of the BDD-based inference algorithm for CPT-L.

The computational cost of the algorithms linear in the size of the BDD graph, which in turn depends strongly on the chosen variable ordering $x_1, \ldots, x_n$. Unfortunately, computing an optimal variable ordering is NP-hard. However, existing implementations of BDD packages [Somenzi, 2009]contain sophisticated heuristics to find a good ordering for a given function efficiently.

4.2.3 Partially Lifted Inference for CPT-Theories

We have so far specified CPT-L theories using first-order logic, but carried out inference at the ground level. This is a common strategy in many statistical relational learning frameworks: the first-order model specification serves as a template language from which a ground model is constructed for inference. A popular approach is to use graphical models as ground models. These can be directed (e.g., Relational Bayesian Networks [Jaeger, 1997], Bayesian Logic Programs [Kersting and De Raedt, 2007], or CP-logic [Vennekens et al., 2006]), or undirected (e.g., Markov Logic Networks [Richardson and Domingos, 2006]).

In CPT-L, the grounded inference problem takes the form of a (propositional) Boolean formula, for which we need to compute all satisfying assignments. This problem can be solved efficiently using binary decision diagrams, as shown in Section 4.2.2. However, the size of the inference problem (and resulting BDD) depends on the size of the grounded first-order model, which can be large compared to the original first-order model specification. Recent work on *lifted inference* in first-order models (see, for example, [Poole, 2003] and [Milch et al., 2008]) has shown that computational efficiency can be improved significantly if inference is performed directly at the first-order level. Exploiting symmetries in the model repeating the same or similar calculations can be avoided. We now discuss a lifted inference algorithm for CPT-theories. The general idea is to solve a part of the overall inference problem directly at the first-order level, without compiling it into the binary decision diagram. The approach is best illustrated using an example:

Example 4.6 *Reconsider the CPT-Theory given in Example 4.2*

$$\begin{aligned} r_1 &= p(X) : 0.2 \vee q(X) : 0.8 \longleftarrow q(X) \\ r_2 &= p(a) : 0.5 \vee (q(b) \wedge q(c)) : 0.5 \longleftarrow \neg q(b) \\ r_3 &= p(X) : 0.7 \vee nil : 0.3 \longleftarrow p(X). \end{aligned}$$

Suppose we want to compute the probability $P(I_{t+1} \mid I_{[0,t]})$*, where* $I_t = \{q(a), q(b), p(1), p(2), p(3)\}$*,* $I_{t+1} = \{p(a), p(b)\}$*, and* $I_{[0,t-1]}$ *are irrelevant as the theory refers only to the previous time-point. Rules* r_1 *and* r_3 *are applicable, and*

$$\mathbf{R}_t = \{r_1[X/a], r_1[X/b], r_3[X/1], r_3[X/2], r_3[X/3]\}.$$

We need to compute

$$P(I_{t+1} \mid I_t) = \sum_{\sigma \in \Gamma} P(\sigma) \ , \tag{4.8}$$

where Γ *is the set of selections yielding* I_{t+1} *from* I_t*. Computing this sum over probabilities of selections* $\sigma \in \Gamma$ *is the ground inference problem, which can be solved using BDDs as explained in Section 4.2.2. According to Equation* (4.1)*, the* $P(\sigma)$ *are of the form*

$$P(\sigma) = f_{11} f_{12} f_{31} f_{32} f_{33} \ ,$$

where $f_{11}, f_{12} \in \{0.2, 0.8\}$ *are the probabilities of selected head elements of ground rules* $r_1[X/a], r_1[X/b] \in \mathbf{R}_t$*, and* $f_{31}, f_{32}, f_{33} \in \{0.7, 0.3\}$ *are the probabilities of selected head elements of ground rules* $r_3[X/a], r_3[X/b], r_3[X/c] \in \mathbf{R}_t$*.*

However, inspecting rule r_1 *and* I_{t+1}*, we see that irrespective of the substitution* θ *grounding rule* r_1 *in* I_t*, only the first head element of* r_1 *can be used in a selection.*

Thus, factors f_{11} *and* f_{12} *are always* 0.2*, and Equation* (4.8) *simplifies to*

$$P(I_{t+1} \mid I_t) = 0.2 \cdot 0.2 \sum_{\sigma' \in \Gamma'} P(\sigma') \ , \tag{4.9}$$

where σ' *only selects head elements for rule* r_3*. That is,* $P(\sigma')$ *is of the form*

$$P(\sigma') = f_{31} f_{32} f_{33}.$$

Note that the remaining ground inference problem—summing over the partial selections σ'*—is smaller than the original one given by Equation* (4.8)*. The remaining problem can be solved using the BDD-based inference method as explained above. However, when converting this inference problem to a Boolean formula* f*, we need to take into account that some facts appearing in the next interpretation* I_{t+1} *have already been generated by the head elements selected for groundings of rule* r_1*, and thus do not need to be generated any more by groundings of rule* r_3*. That is, we simply ignore already generated facts in Step 4 of the construction of* f*.*

In fact, we can go one step further, and note that also for rule r_3 *we can determine the selected head element irrespective of the substitution used to ground the rule in* I_t*. It is easily determined by logical inference that head element* $p(X)$ *cannot be grounded in* I_{t+1} *given that the body* $p(X)$ *is grounded in* I_t*, thus only the second head element can be used for any grounding of rule* r_3 *in any selection* σ'*. Thus, Equation* (4.9) *is further simplified to*

$$\begin{aligned} P(I_{t+1} \mid I_t) &= 0.2 \cdot 0.2 \cdot 0.3 \cdot 0.3 \cdot 0.3 \\ &= 0.2^{K_{r_1}} 0.3^{K_{r_3}} \ , \end{aligned}$$

where K_{r_i} *is the number of groundings of rule* r_i *in* I_t*.*

The key observation in the above example is that for both r_1 and r_3 we could logically infer the head element used in any selection $\sigma \in \Gamma$ under any grounding of the rules in I_t. Note that in general, only a subset of the rules can be removed from the ground inference problem in this way.

Generalizing from Example 4.6, we can describe the partially lifted inference algorithm for any given CPT-theory $\mathcal{T} = \{r_1, \ldots, r_k\}$ and inference problem $P(I_{t+1} \mid I_{[0,t]})$ as follows:

1. Let $\mathbf{R}_t$ denote the set of all ground rules applicable in $I_{[0,t]}$
2. Define

$$\begin{aligned} \overline{\mathbf{R}}_t = \{ r\theta \in \mathbf{R}_t \mid & I_{t+1},\ I_{[0,t]} \text{ logically determines} \\ & \text{the head element selected for } r\theta \} \end{aligned}$$

For a rule $r\theta \in \overline{\mathbf{R}}_t$, let $\overline{\sigma}(r\theta)$ denote the head element that must be selected.

3. Compute

$$P(I_{t+1} \mid I_{[0,t]}) = \prod_{r\theta \in \overline{\mathbf{R}}_t} P(\overline{\sigma}(r\theta) \mid r\theta) \sum_{\sigma' \in \Gamma'} P(\sigma')$$

$$= \prod_{r \in \mathcal{T}} \prod_{c_r \in head(r)} P(c_r \mid r)^{K_{r,c}} \sum_{\sigma' \in \Gamma'} P(\sigma'), \tag{4.10}$$

where

$$K_{r,c} = |\{r\theta \in \overline{\mathbf{R}}_t \mid \overline{\sigma}(r\theta) = c\theta\}|$$

and Γ' is the set of selections of head elements for rules in $\mathbf{R}_t \setminus \overline{\mathbf{R}}_t$ that yield I_{t+1} from $I_{[0,t]}$, given that we select head element $\overline{\sigma}(r\theta)$ for rules in $r\theta \in \overline{\mathbf{R}}_t$. Note that in Equation (4.10) we have integrated all ground rules for which a particular head element c_r has to be selected into one factor, which has to be taken to the power of $K_{r,c}$, namely the number of such ground rules. Thus, we have performed a partially lifted probability calculation.

The set $\overline{\mathbf{R}}_t$ contains those grounded rules $r\theta$ for which we can prove — using logical inference on $body(r\theta)$, $head(r\theta)$, and the interpretations $I_{[0,t]}$ and I_{t+1} — that a particular head element $\overline{\sigma}(r\theta)$ has to be selected for $r\theta$. For instance, all groundings of rule r_1 in Example 4.6 are in this set, because no ground facts of the form $q(X)\theta$ appear in I_{t+1}, and thus the first head element of r_1 always has to be selected. In fact, for Example 4.6 we have $\overline{\mathbf{R}}_t = \mathbf{R}_t$. The term $K_{r,c}$ is the number of groundings of a rule $r \in \mathcal{T}$ for which we know that the head element $c \in head(r)$ is selected for the grounded rule. For instance, in Example 4.6, $K_{r_1,p(X)} = 2$ and $K_{r_3,nil} = 3$. In practice, the counting variables $K_{r,c}$ can be computed as follows. For each rule r, we first determine the set of groundings θ such that $r\theta$ holds in $I_{[0,t]}$ and exactly one of the grounded head elements holds in I_{t+1}; this can be achieved with a single logical query. We then count for each head element $c_r \in head(r)$ the number of times the unique grounded head determined in the first step was subsumed by c_r, this yields the term $K_{r,c}$.

Comparing the outlined partially lifted inference algorithm to other lifted inference algorithms proposed in the literature, such as first-order probabilistic inference [Poole, 2003] or lifted inference with counting formulas [Milch et al., 2008], we note that it is much simpler and, correspondingly, more limited in scope. Nevertheless, it proved surprisingly effective in our experimental evaluation (see Section 4.3).

Note that the efficiency of the presented inference algorithm depends on the fact that the selection of a particular head element is enforced by a given successor interpretation. This in turn depends on the closed-world assumption, which states that any atom not observed is false.

4.2.4 Parameter Estimation

Assume the structure of a CPT-theory is given, that is, a set $\mathcal{T}(\pi) = \{r_1, \ldots, r_k\}$ of rules of the form

$$r_i = (c_{i1} : p_{i1}) \vee \ldots \vee (c_{in_i} : p_{in_i}) \longleftarrow b_{i1}, \ldots, b_{im_i},$$

where $\pi = \{p_{ij}\}_{i,j}$ are the unknown parameters to be estimated from a set of training sequences $\mathcal{D}$. A standard approach is to find maximum-likelihood parameters

$$\pi^* = \arg\max_{\pi} P(\mathcal{D} \mid \mathcal{T}(\pi)),$$

that is, to set the parameters such that we maximize the probability of generating the data $\mathcal{D}$ from $\mathcal{T}$. When generating $\mathcal{D}$ from $\mathcal{T}$, a rule $r_i \in \mathcal{T}$ is typically applied multiple times: in the form of different groundings $r_i\theta$, and in different transitions (appearing in different training sequences). We would like to set

$$\forall i, j: \quad p_{ij} = \frac{\kappa_{ij}}{\sum_{l=1}^{n_i} \kappa_{il}}, \tag{4.11}$$

where κ_{ij} denotes the number of times head element c_{ij} was selected in any application of the rule r_i while generating $\mathcal{D}$. However, the quantity κ_{ij} is not directly observable. To see why this is so, first consider a single transition $I_{[0,t]} \rightarrow I_{t+1}$ in one training sequence. We know the set of rules $\mathbf{R}_t$ applied in the transition; however, there are in general many possible selections σ of rule head elements yielding I_{t+1}. The information about which selection was used, that is, which rule has generated which fact in I_{t+1}, is hidden. We will now derive an efficient Expectation-Maximization algorithm in which the unobserved variables are the selections used at a transition, and κ_{ij} the sufficient statistics. To this aim, we first need to compute expected values of the κ_{ij} given the observations and the current model parameters π, and then re-estimate π according to Equation (4.11) where the κ_{ij} are replaced by their expectation.

To keep the notation uncluttered, we first consider a single transition $\Delta = I_{[0,t]} \rightarrow I_{t+1}$. Let $\mathbf{R}_t$ denote the set of rules applicable in the transition, and let $\kappa_{ij}^{\theta} \in \{0, 1\}$ denote whether the grounded head element $c_{ij}\theta$ was selected in the application of a grounded rule $r_i\theta \in \mathbf{R}_t$. Let furthermore $\Gamma = \{\sigma \mid I_{[0,t]} \xrightarrow{\sigma} I_{t+1}\}$ be the set of selections yielding I_{t+1}. For a given selection $\sigma \in \Gamma$, we have

$$\kappa_{ij}^{\theta} = \begin{cases} 1: & \sigma(r_i\theta) = c_{ij}\theta \\ 0: & otherwise \ , \end{cases} \tag{4.12}$$

and

$$\kappa_{ij} = \sum_{\theta : r_i\theta \in \mathbf{R}_t} \kappa_{ij}^{\theta} \tag{4.13}$$

where the sum runs over all groundings $r_i\theta \in \mathbf{R}_t$ of rule r_i. However, the selection σ is not observed, thus we instead have to consider the expectation $\mathbb{E}[\kappa_{ij} \mid \pi, \Delta]$ of κ_{ij} with respect to the posterior distribution $P(\sigma \mid \pi, \Delta)$ over selections given the data and current parameters. It holds that

$$\begin{aligned}\mathbb{E}[\kappa_{ij}^{\theta} \mid \pi, \Delta] &= P(\kappa_{ij}^{\theta} = 1 \mid \pi, \Delta) \\ &= \sum_{\sigma \in \Gamma} P(\kappa_{ij}^{\theta} = 1 \mid \sigma) P(\sigma \mid \pi, \Delta) \ , \end{aligned} \tag{4.14}$$

where $P(\kappa_{ij}^{\theta} = 1 \mid \sigma) \in \{0, 1\}$ according to Equation (4.12). Equation (4.13) now implies

$$\mathbb{E}[\kappa_{ij} \mid \pi, \Delta] = \sum_{\theta : r_i\theta \in \mathbf{R}_t} \mathbb{E}[\kappa_{ij}^{\theta} \mid \pi, \Delta],$$

which concludes the expectation step of the Expectation-Maximization algorithm for a single transition Δ. If the data $\mathcal{D}$ contains multiple transitions (possibly appearing in multiple sequences), we can simply sum up the quantities $\mathbb{E}[\kappa_{ij} \mid \pi, \Delta]$ for each transition. Finally, given the expectation of the sufficient statistics κ_{ij}, the maximization step in EM is

$$p_{ij}^{(new)} = \frac{\mathbb{E}[\kappa_{ij} \mid \pi, \mathcal{D}]}{\sum_j \mathbb{E}[\kappa_{ij} \mid \pi, \mathcal{D}]}.$$

As usual, expectation and maximization steps are iterated until convergence in likelihood space.

The key algorithmic challenge in the outlined EM algorithm is to compute the expectation given by Equation (4.14) efficiently. Note that this again involves summing over all selections yielding the next interpretation, much as in the inference problem discussed in Sections 4.2.2 and 4.2.3. In fact, the quantity $\mathbb{E}[\kappa_{ij}^{\theta} \mid \pi, \Delta]$ can also be obtained from the upward and downward probabilities introduced in Section 4.2.2. More formally, the following holds:

Theorem 4.3 *Let p_{ij} be the parameter associated with head element c_{ij} in rule r_i, let $\Delta = I_{[0,t]} \to I_{t+1}$ be a single transition, and let $r_i\theta \in \mathbf{R}_t$ denote a grounding of r_i applicable in $I_{[0,t]}$. Let $N_1, \ldots, N_k$ be all nodes in the BDD associated with the Boolean variable $r_i\theta.c_{ij}\theta$ resulting from the grounded rule $r_i\theta$, and let N_l^+ be the child on the positive branch of N_l. Then*

$$\mathbb{E}[\kappa_{ij}^{\theta} \mid \pi, \Delta] = \frac{1}{P(I_{t+1} \mid I_{[0,t]})} \sum_{l=1}^{k} \beta(N_l) p_{ij} \alpha(N_l^+). \tag{4.15}$$

As for the inference problem discussed in Section 4.2.2, we can thus compute the estimation step given by Equation (4.14) in time, linear in the size of the BDD. The theorem can be proven using similar techniques as in the proof of Theorem 4.2; however, the proof is slightly more involved and thus moved to Appendix 8.3.

Finally, note that at this point we can again make use of the partial lifted inference algorithm discussed in Section 4.2.3. A part of the expectation computation is then solved directly at the first-order level, while the rest is solved using dynamic programming in the BDD as explained above.

Note that the presented algorithms for inference and parameter estimation can be significantly more efficient than the corresponding algorithms in the more general CP-logic framework. Specifically, in CP-logic the inference and learning problems are typically grounded into a Bayesian network, which can grow very large depending on the characteristics of the domain. This often makes (exact) inference computationally challenging. In contrast, the inference and learning techniques we discussed here take advantage of the particular problem setting and model structure (that is, sequential and fully observable data). The experimental evaluation presented in Section 4.3 indeed shows that with these techniques we can perform exact inference in only seconds, for problems where the ground Bayesian network would contain hundreds of thousands of nodes. In the next Chapter 5 we will study an algorithm similar to the one presented here, which allows one to learn parameters of ProbLog programs.

4.2.5 Prediction

Assume we are given an observation sequence $I_0, \ldots, I_t$, a CPT-theory $\mathcal{T}$, and a property of interest F (represented as a first-order query), and would like to compute $P(I_{t+d} \models F \mid I_0, \ldots, I_t, \mathcal{T})$. For instance, a robot might like to know the probability that a certain world state is reached at time $t + d$, given its current world model and observation history. Or, in the MMOG domain, we might want to compute the probability that a particular player will be the winner at time $t + d$, given a model of game dynamics and an observation history. We will assume that F is any first-order query that could be posed to a logic programming system such as Prolog, making use of the available background knowledge B.

Powerful statistical relational learning systems are in principle able to compute the quantity $P(I_{t+d} \models F \mid I_0, \ldots, I_t, \mathcal{T})$ exactly by "unrolling" the world model into a large dynamic graphical model. However, this is computationally expensive as it requires to marginalize out all (unobserved) intermediate world states $I_{t+1}, \ldots, I_{t+d-1}$, and thus often not practical in complex worlds. In contrast, inference in CPT-theories draws its efficiency from the full observability assumption, as outlined in Section 4.2. As an alternative to the "unrolling" approach, we thus propose a straightforward sample-based approximation to compute $P(I_{t+d} \models F \mid I_t, \mathcal{T})$ that

preserves the efficiency of our approach. The idea is to obtain independent samples from the Boolean random variable $I_{t+d} \models F$ given $\mathcal{T}$ and $I_0, \ldots, I_t$, and estimate the desired probability as the fraction of positive samples.

Given $I_0, \ldots, I_t$, it is straightforward to obtain independent samples of the conditional distribution $P(I_{t+1}, \ldots, I_{t+d} \mid I_0, \ldots, I_t, \mathcal{T})$ by forward sampling from the stochastic process represented by $\mathcal{T}$, as explained in Section 4.2.1. Ignoring $I_{t+1}, \ldots, I_{t+d-1}$, we can simply check whether $I_{t+d} \models F$ in the sampled interpretation I_{t+d}. After repeatedly sampling interpretations $I_{t+d}^{(1)}, \ldots, I_{t+d}^{(K)}$ in this fashion, the fraction of $I_{t+d}^{(k)}$ for which $I_{t+d}^{(k)} \models F$ is then an unbiased estimator of the true probability $P(I_{t+d} \models F \mid I_t, \mathcal{T})$, and will in fact quickly converge towards this true probability for large K.

4.2.6 Filtering

The algorithm introduced in the previous sections estimates the probability that a query holds in the future given a fully observable past. A related task is to calculate the probability of query if the past is only partially observable. This is commonly referred to as the filtering distribution. Given a sequence of hidden states $I_0, \ldots, I_t$ the starting states, a sequence of observations $O_0, \ldots, O_{t+d}$, a HCPT-theory $\mathcal{T}$, and a property of interest F (represented as a first-order query), the filtering distribution is the probability $P(I_{t+k} \models F \mid I_0, \ldots, I_t, O_0, \ldots, O_{t+k}, \mathcal{T})$.

$$P(I_{t+k} \models F \mid I_0, \ldots, I_t, O_0, \ldots, O_{t+k}, \mathcal{T}) = \sum_{I_{t+k} \models F} P(I_{t+k} \mid I_0, \ldots, I_t, O_0, \ldots, O_{t+k}, \mathcal{T}) \quad (4.16)$$

Example 4.7 *The property of interest F can be a query, for example, $a(P1, Act)$, $a(P2, Act)$, $P1 \neq P2$ whether* two persons performing the same activity *in the model from example 4.4.*

Exact calculation of this distribution is typically prohibitively slow due to the large state space [Boyen and Koller, 1998]. The prediction algorithm described in the previous section can be utilized to approximate the filtering distribution using rejection sampling. The idea is to sample a set of sequences, where the sequences consistent with observations can be used as empirical filtering distribution. However, in practice probably only a few samples would be consistent with the observations and would therefore require a large amount of samples until a reasonably good estimated is obtained. This is commonly known as the rejection problem.

Therefore we use sampling importance resampling (SIR) [Doucet et al., 2001]: we sample from a proposal distribution and compute importance weights that make

up for the difference between the two distributions. The proposal distribution is chosen so that the fraction of rejected samples is low.

In the following we first briefly discuss the mechanics of SIR. Afterwards we alter the original CPT-L inference algorithm (Section 4.2.2) using a BDD to represent the distributions required by SIR. Finally we give the algorithm to sample states from this distribution using the constructed BDD.

Sampling Importance Resampling (SIR) The filtering distribution can be approximated by a set of N particles (w_k^i, I_k^i) consisting of a weight w_k^i and a state I_k^i, where the index $i < N$ indicates the particles. The weights are an approximation of the relative posterior distribution. The empirical distribution is defined as

$$\hat{p}(I_k|O_{[1,k]}) = \frac{1}{N}\sum_{i=1}^{N} w_i \delta_{I_k^i}(I_k),$$

where $\delta_{I_k^i}(\cdot)$ is the point mass distribution located at I_k^i.

The SIR algorithm calculates the samples iteratively. A single step is described in Algorithm 3. In each step, the particles are drawn from a **proposal distribution** $\pi(I_k|I^i_{[0,k-1]}, O_{0:k})$ and each particle's weight is calculated. In principle any admissible distribution can be chosen as proposal distribution. A distribution is called **admissible** if it has probability greater than zero for each state, that has probability greater than zero for the target distribution. So the correctness does not depend on the choice, but on the sample variance, and thus the required number of particles, largely depends on this choice.

Typical sampling distributions are the transition prior $P(I_k|I^i_{k-1})$, a fixed importance function $\pi(I_k|I^i_{[0,k-1]}, O_{0:k}) = \pi(I_k)$, or the transition posterior $p(I_k|I^i_{[0.k-1]}, O_{[0,k]})$.

Optimal Proposal distribution The transition prior $p(I_k|I_{k-1})$ is often used as proposal distribution for SIR as it allows for efficient sampling. Using the transition prior means, on the other hand, that the state space is explored without any knowledge of the observations which makes the algorithm sensitive to outliers. While this nonetheless works well in many cases, it is problematic in discrete, high dimensional state spaces when combined with spiked observation distributions. But high dimensional state spaces are common, especially in relational domains. It can be shown that the proposal distribution $p(I_k^i|I_{k-1}^i, O_k)$[1] together with weight update $w_k^i := w_{k-1}^i P(O_k|I_{k-1}^i)$ is optimal [Doucet et al., 2001] and does not suffer from this problem.

[1]Here it is crucial, to realize that the observation O_k is the observation generated by the state I_k.

Algorithm 3 Sample a set of n particles N, for the k-th time step using the proposal distribution π

function SAMPLE$(\pi, (w_k^i, I^i_{[0.k-1]})_i, O_{1:k}, \mathcal{T})$
  for $i = 1, \ldots, N$ **do**
    $I_k^i \sim \pi(I_k | I^i_{[0.k-1]}, O_{1,k})$
    $\hat{w}_k^i := w_{k-1}^i \frac{P_O(O_k|O_k^i) P_s(I_k^i | I_{k-1}^i)}{\pi(I_k^i | I^i_{[0.k-1]}, O_{[0.k]})}$
  Normalize weights $w_k^i = \hat{w}_k^i / \sum_j \hat{w}_k^j$
  if $\hat{N}_{thresh} > \overbrace{\left(\sum_i (w_k^i)^2\right)^{-1}}^{\text{effective \# particles}}$ **then**
    Sub-sample N particles of I_k^i according to w_k^i with weight $1/P$

BDD construction: To sample from the proposal distribution and update the weight efficiently we build a BDD that represents $P(O_k | I_{k-1}^i)$. The algorithm is a modification of the algorithm presented in Section 4.2.2.

1. Initialize $f := true$, $I_{max} = \varnothing$ will be the "maximal" successor state
2. Let $\mathbf{R}_t$ denote the set of unobservable, ground rules which are applicable in $I_{[0,t]}$. Rules $r \in \mathbf{R}_t$ are of the form $r = c_1 : p_1, \ldots, c_n : p_n \longleftarrow b_1, \ldots, b_m$, where c_i are conjunctions of literals (see Definition 4.1) which are not observable.
3. For all rules $r \in \mathbf{R}_t$ do:
   (a) $f := f \wedge (r.c_1 \vee \ldots \vee r.c_n)$, where $r.c_i$ denotes a new (propositional) Boolean variable whose unique name is the concatenation of the name of the rule r with the head element c_i.
   (b) $f := f \wedge (\neg r.c_i \vee \neg r.c_j)$ for all $i \neq j$
   (c) $h_i \leftarrow r.h_i$; $I_{max} = I_{max} \cup h_i$
4. Let $\mathbf{S}_t$ denote the set of all observable ground rules, which are applicable in I_{max}. Rules $r \in \mathbf{R}_t$ are of the form $r = c_1 : p_1, \ldots, c_n : p_n \longleftarrow b_1, \ldots, b_m$, where c_i are conjunctions of literals (see Definition 4.1) which are observable.
5. for all observations $(r = (p_1 : h_1, ..., p_n : h_n) \leftarrow b_1, ..., b_m)$ in $\mathbf{S}_k$
   (a) $f := f \wedge ((r.h_1 \vee ... \vee r.h_n) \leftrightarrow (b_1, \ldots, b_n))$, for $h_i \in O_k$
   (b) $f := f \wedge (\neg r.h_i \vee \neg r.h_j)$ for all $i \neq j$
6. For all facts $l \in I_{O_k}$

(a) Initialize $g := false$

(b) for all $r \in \mathbf{S}_k$ with $p : l \in head(r)$ do $g := g \vee r.l$

(c) $f := f \wedge g$

The algorithm builds a BDD representation of a formula which computes the joint probability of all possible selections that result in a transition for which the following four conditions hold. The transition (a) starts at I^i_{t-1} and (b) goes over to a valid successor state I_t. In I_t it (c) generates the observation O_t using (d) the observable rule applicable in I_t. Each node of the generated BDD $r : h_i$ corresponds to selecting (for one rule) the head h_i or not, as dictated by the probability p_i. The similarities to the previous BDD constructions becomes obvious, when comparing the lines up to 3.b and the last lines of both algorithms.

BDD sampling Sampling a path according to the p_i from the root of this BDD to the terminal node with label 1 corresponds to sampling a value from $p(I_k|I^i_{k-1}, O_k)$. However, in most cases, sampling paths naïvely according to the p_i's will yield a path ending in the 0-terminal, that will then have to be rejected. Notably this would correspond to rejection sampling. Therefore, at every node when choosing the corresponding sub-tree, we base our choice not only on its probability, but also on the probability of reaching the 1-terminal through this sub-tree. This corresponds to conditioning the paths such that we get a valid successor state together with the observation O_k. This corresponds to sampling at every node from the normalized upward probability as defined in Section 4.2.2 and Figure 4.5.

The algorithm starts by calculating for every node N the upward probabilities $\alpha(N)$. Afterwards a single path is sampled. The positive branch is chosen in every node according to

$$\frac{p_N \cdot \alpha(N^+)}{\alpha(N^-) + p_N \cdot \alpha(N^+)} = \frac{p_N \cdot \alpha(N^+)}{\alpha(N)},$$

where N^+ denotes the child on the high and N^- the child on the low branch. The next state consists of the set of unobservable literals, where a node representing it is left towards the positive branch.

4.3 Experimental Evaluation

In this section, we experimentally validate the proposed CPT-L approach in several (artificial and real-world) domains as well as in different learning settings. The general setting discussed in this chapter, namely constructing models for

stochastic processes with complex state representations, covers a wide range of application domains. It is appropriate whenever systems evolve over time and are complex enough that their states cannot easily be described using a propositional representation. A prominent example is states that are characterized by a graph structure relating different agents and/or world artifacts at a given point in time (as in dynamic social networks, computer networks, the world wide web, games, marketplaces, et cetera). In this setting, observations consist of sequences of labeled (hyper)graphs, cf. Figure 4.10. To experimentally evaluate CPT-L, we have selected the following domains as representative examples:

Stochastic Blocks World Domain This domain is a stochastic version of the well-known artificial *blocks world* domain, representing an agent that is moving blocks which are stacked on a table. We use this artificial domain to perform controlled experiments, testing the scaling and convergence behavior of the inference and learning algorithms.

Chat Room Domain This domain is concerned with the analysis of user interaction in chat rooms. We have monitored a number of IRC chat rooms in real time, and recorded who was sending messages to whom using the PieSpy utility [Mutton, 2004]. This results in dynamically changing graphs of user interaction, representing the social network structure among chat room participants, cf. Figure 4.7. We learn these dynamics using separate models for different chat rooms. The resulting set of models can be used to visualize commonalities and differences in the behavior displayed in different chat rooms, thereby characterizing the underlying user communities.

Massively Multiplayer Online Game Domain As a third evaluation domain introduced by Thon et al. [2008], we consider the large-scale, massively multiplayer online strategy game Travian.[2] Game worlds feature thousands of players, game artifacts such as cities, armies, and resources, and social player interaction in alliances. Game states in Travian are complex and richly structured, and transitions between game states highly stochastic as they are determined by player actions. We have logged the state of a "live" game server over several months, recording high-level game states as visualized in Figure 4.10. We address different learning tasks in the Travian domain, such as predicting player actions (prediction setting) and identifying groups of cooperating alliances (classification setting).

Activity recognition domain The final evaluation domain is inspired by the model defined by Biswas et al. [2007]. There a person is observed during writing, typing, using a mouse, eating, and so on. The task in this domain is to infer the activity performed by a person based on observations about which objects he is using. We used the set of rules defined in the original paper, but used random parameters because the original dataset is not available. Also, the semantics of the rules is slightly changed as due to the closed world assumption applied in CPT-L.

[2] `www.travian.com;www.traviangames.com`

The goal of our experimental study is two-fold. First, we want to evaluate the effectiveness of the proposed approach (Q1). That is, we explore whether it is possible to learn dynamic stochastic models for the above-mentioned relational domains, and to solve the resulting inference, prediction, and classification tasks. Our second goal is to evaluate the efficiency of the proposed algorithms (Q2). That is, we will evaluate the scaling behavior for domains with a large number of objects and relationships, and in particular explore the advantage of performing partially lifted inference in such domains. Experiments to address these questions will be presented in turn for the three outlined evaluation domains in the rest of this section.

4.3.1 Experiments in The Stochastic Blocks World Domain

As an artificial testbed for CPT-L, we performed experiments in a stochastic version of the well-known *blocks world* domain. The domain was chosen because it is truly relational and also serves as a popular artificial world model in agent-based approaches such as planning and reinforcement learning. Moreover, application scenarios involving agents that act and learn in an environment are one of the main motivations for CPT-L.

World Model The blocks world we consider consists of a table and a number of blocks. Every block rests on exactly one other block or the table, denoted by a fact $on(A, B)$. Blocks come in different sizes, denoted by $size\_of(A, N)$ with $N \in \{1, \ldots, 4\}$. A predicate $free(B) \longleftarrow not(on(A, B))$ is defined in the background knowledge. Additionally, a background predicate $stack(A, S)$ defines that block A is part of a stack of blocks, which is represented by its lowest block S. Actions in the blocks world domain are of the form $move(A, B)$. If both A and B are free, the action moves block A on B with probability $1 - \epsilon$, with probability ϵ the world state does not change. Furthermore, a stack S can start to jiggle, represented by $jiggle(S)$. A stack can start to jiggle if its top block is lifted, or a new block is added to it. Furthermore, stacks can start jiggling without interference from the agent, which is more likely if they contain many blocks and large blocks are stacked on top of smaller ones. Stacks that jiggle collapse in the next time step, and all their blocks fall on the table. Two example rules from this domain are

$$(jiggle(S) : 0.2) \vee (nil : 0.8) \longleftarrow move(A, B), stack(A, S)$$

$$(jiggle(S) : 0.2) \vee (nil : 0.8) \longleftarrow move(A, B), stack(B, S),$$

they describe that stacks can start to jiggle if blocks are added to or taken from a stack. Furthermore, we assume the agent follows a simple policy that tries to build a large stack of blocks by repeatedly stacking the free block with second-lowest ID on the free block with lowest ID. This strategy would result in one large stack of

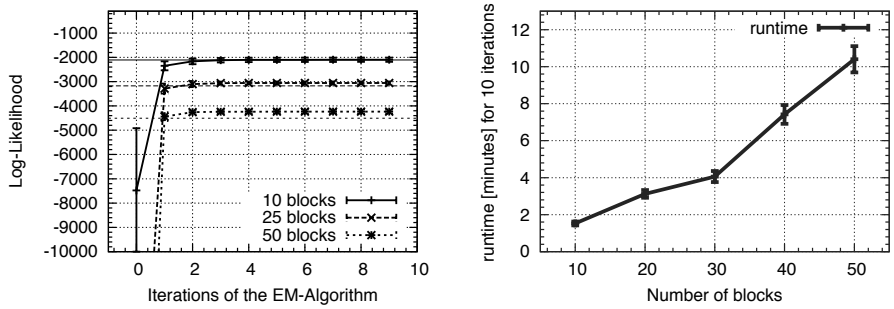


Figure 4.6: Left graph: per-sequence log-likelihood on the training data as a function of the EM iteration. Right graph: Running time of EM as a function of the number of blocks in the world model.

blocks if stacks never collapsed. In our experiments, the policy was supplied as background knowledge, that is, the predicate *move*/2 was hard-coded by a logical definition in the background knowledge and not part of the learning problem. The model had 14 rules with 24 parameters in total.

Results in The Blocks-World Domain In a first experiment, we explore the convergence behavior of the EM algorithm for CPT-L. The world model together with the policy for the agent, that specifies which block to stack next, is implemented by a (gold-standard) CPT-theory $\mathcal{T}$, and a training set of 20 sequences of length 50 each is sampled from $\mathcal{T}$. From this data, the parameters are re-learned using EM. Figure 4.6, left graph, shows the convergence behavior of the algorithm on the training data for different numbers of blocks in the domain, averaged over 15 runs. It shows rapid and reliable convergence. Figure 4.6, right graph, shows the running time of EM as a function of the number of blocks. The scaling behavior is roughly linear, indicating that the model scales well to reasonably large domains. Absolute running times are also low, with about 1 minute for an EM iteration in a world with 50 blocks[3]. The theory learned (Figure 4.6) is very close to the ground truth ("gold standard model") from which training sequences were generated. On an independent test set (also sampled from the ground truth), log-likelihood for the gold standard model is -4510.7, for the learned model it is -4513.8, while for a theory with randomly initialized parameters it is -55999.4 (50 blocks setting).

[3]All experiments were run on standard PC hardware, 2.4GHz Intel Core 2 Duo processor, 1GB memory.

Manual inspection of the learned model also shows that parameter values are on average very close to those in the gold-standard model.

The experiments presented so far show that relational stochastic domains of substantial size can be represented in CPT-L. The presented algorithms scale well in the size of the domain, and show robust convergence behavior.

4.3.2 Experiments in The Chat Room Domain

For our experiments in the chat room domain, we have selected the following 7 well-frequented IRC chat rooms: *football@irc.efnet.net*, *iphone@irc.efnet.net*, *computer@irc.efnet.net*, *poker@irc.efnet.net*, *math@irc.efnet.net*, *politics@irc.efnet.net*, and *travian@irc.travian.org*. Each chat room was monitored for one day using the PieSpy utility [Mutton, 2004], generating a sequence of user interaction graphs as those shown in Figure 4.7. For each chat room, we selected the first 100 observations in the sequence of user interaction graphs as a single observation sequence for that chat room, yielding 7 observation sequences $S_1, \ldots, S_7$.

We have again hand-coded a simple CPT-theory $\mathcal{T}$ for this domain, which makes use of a number of graph-theoretic properties defined in the background knowledge, such as graph centrality, node degree, closeness, betweenness, and co-citation. As an example rule, consider

$$communicates(P1, P2) : 0.1 \vee nil : 0.9 \longleftarrow cocitation(P1, P2, CC),$$
$$\neg communicates(P1, P2), \neg communicates(P2, P1)$$

encoding that two chat participants start talking to each other if there is a third participant with whom they have both talked before. The following three rules encode that a random person starts to communicate with another person which

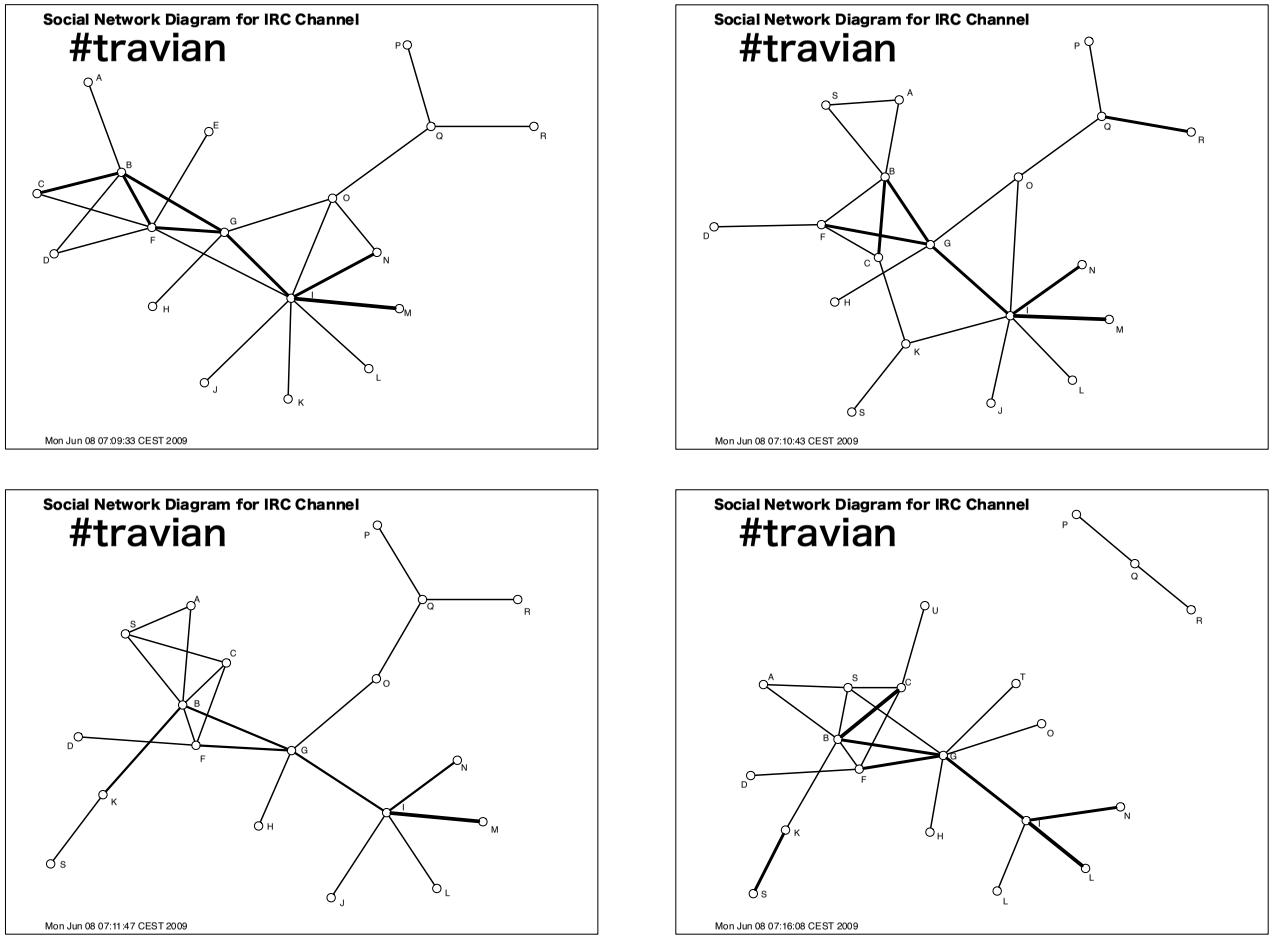


Figure 4.7: User interaction graphs from the Chat Room Domain. Shown are four different time points during the observation sequence recorded for the *irc.travian.org* chat room.

has above average *betweeness*, *degree*, or *closeness*.

$$\begin{aligned}
&communicates(P1, P2) : 0.1 \vee nil : 0.9 \longleftarrow betweeness(P1, C1),\\
&\qquad\qquad avg_betweeness(Avg), C1 > Avg,\\
&\qquad\qquad \neg communicates(P1, P2), \neg communicates(P2, P1).\\
&communicates(P1, P2) : 0.1 \vee nil : 0.9 \longleftarrow degree(P1, C1), person(P2),\\
&\qquad\qquad avg_degree(Avg), C1 > Avg,\\
&\qquad\qquad \neg communicates(P1, P2), \neg communicates(P2, P1).\\
&communicates(P1, P2) : 0.1 \vee nil : 0.9 \longleftarrow closeness(P1, C1), person(P2),\\
&\qquad\qquad avg_closeness(Avg), person(P1), C1 > Avg,\\
&\qquad\qquad \neg communicates(P1, P2), \neg communicates(P2, P1).
\end{aligned}$$

In the model definition, rule heads also contain a third head element $communicates(P2, P1)$ which signifies communication in the opposite direction. This was omitted above for increased readability. In total the model had 7 rules with 11 parameters (note that a rule with three head elements has two parameters, as parameters must sum to one).

For each chat room, we learn the parameters of the CPT-theory $\mathcal{T}$ using the EM algorithm presented in Section 4.2.4, resulting in 7 CPT-theories $\mathcal{T}_1, \ldots, \mathcal{T}_7$ with the same rule structure but different parameters. Learning took about 10 seconds per theory $\mathcal{T}_i$. The learned CPT-theories can be seen as a probabilistic representation of the typical interaction behavior among members of that chat room, reflecting the corresponding different user communities. For instance, they could represent how quickly the interaction graph changes, the degree of connectivity in the interaction graph, or how large the fluctuation in chat participants is over time. The goal of our experiment is to visualize the commonalities and differences in the behavior of these different user groups. To this end, we have evaluated the likelihood $P(S_i \mid \mathcal{T}_j)$ of each sequence S_i under the learned CPT-theory $\mathcal{T}_j$. This gives an indication as to how well the behavior in chat room i is explained by the model learned for chat room j, thus indicating the similarity in user behavior for the corresponding two communities.

The result of this experiment is visualized in Figure 4.8. We can distinguish different clusters of chat rooms, or, equivalently, user communities. For instance, chat rooms that are concerned with recreational topics such as *travian@irc.travian.org* and

football@irc.efnet.net (as well as *iphone@irc.efnet.net*) are clearly distinguishable from chat rooms concerned with more "serious" topics such as *math@irc.efnet.net* and *politics@irc.efnet.net*.[4] Manual inspection of the learned rule parameters showed that in the "serious" chat domains the likelihood of a communication between two persons mostly depends on the betweenness and degrees of the nodes involved, while in the "recreational" chats shared cocitations are more important.

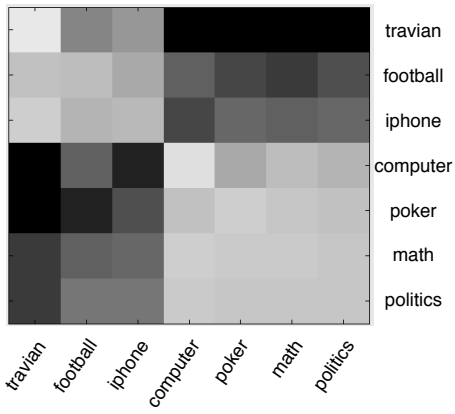


Figure 4.8: Plot of the likelihood $P(S_i \mid \mathcal{T}_j)$ of a sequence S_i (corresponding to chat room i) under the CPT-theory $\mathcal{T}_j$ (learned on chat room j). Rows correspond to models $\mathcal{T}_j$ and columns to sequences S_i. Lighter colors indicate higher likelihoods.

4.3.3 Experiments in The Massively Multiplayer Online Game Domain

The last set of experiments for CPT-L were performed in the *Travian* domain In Travian, players are spread over several independent *game worlds*, with approximately 20.000–30.000 players interacting in a single world. Travian game play follows a classical strategy game setup. A game world consists of a large *grid-map*, and each player starts with a single *city* located on a particular tile of the map. During the course of the game, players harvest *resources* from the environment, improve their cities by construction of *buildings* or research of *technologies*, or found new cities on other (free) tiles of the map. Additionally, players can build different military units which can be used to attack and conquer other cities on the map, or trade resources on a global marketplace.

[4] We use the term "serious" for chat rooms in which participants seem to try to acquire information, e.g., poker is "serious" as the participants are typically trying to learn about different strategies. In recreational chats people are typically interested in the conversation as such.

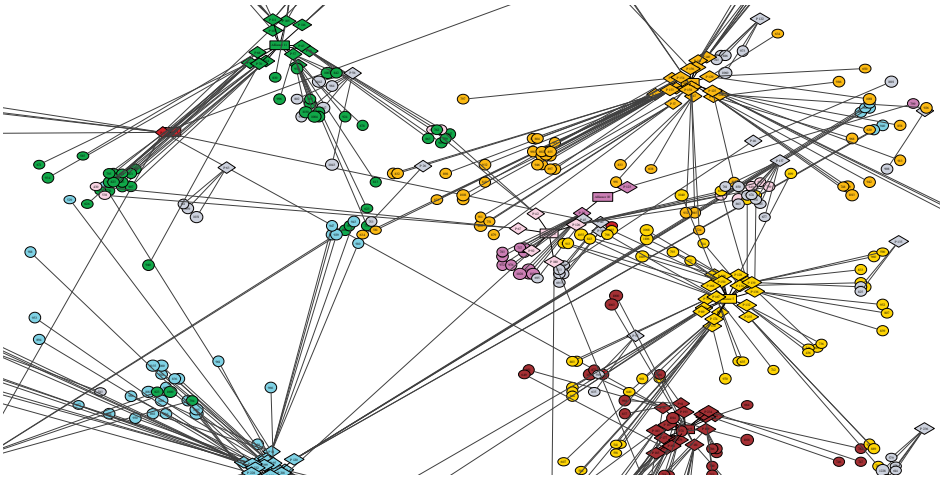


Figure 4.9: High-level view of a (partial) game world in Travian. Circular nodes indicate cities, shown in their true positions on the game's grid-map. Diamond-shaped nodes indicate players, and are connected to all cities currently owned by the player. Rectangular nodes indicate alliances, and are connected to all players currently members of the alliance. (The alliance affiliation is additionally indicated by color-coding of the cities and players.)

In addition to these low-level game play elements, there are high-level aspects of game play involving *multiple players*, which need to cooperate and coordinate their playing to achieve otherwise unattainable game goals. More specifically, in Travian players dynamically organize themselves into *alliances*, for the purpose of jointly attacking and defending, trading resources or giving advice to inexperienced players. Such alliances constitute social networks for the players involved, where diplomacy is used to settle conflicts of interests and players compete for an influential role in the alliance. In the following, we will take a high-level view of the game and focus on modeling player interaction and cooperation in alliances rather than low-level game elements such as resources, troops and buildings. Figure 4.9 shows such a high-level view of a (partial) Travian game world, represented as a graph structure relating cities, players and alliances which we will refer to as a *game graph*. It shows that players in one alliance are typically concentrated in one area of the map—traveling over the map takes time, and thus there is little interaction between players far away from each other.

We are interested in the *dynamic* aspect of this world: as players are acting in the game environment (e.g., by conquering other players' cities and joining or leaving alliances), the game graph will continuously change, and thereby reflect changes in the social network structure of the game. As an example of such transition

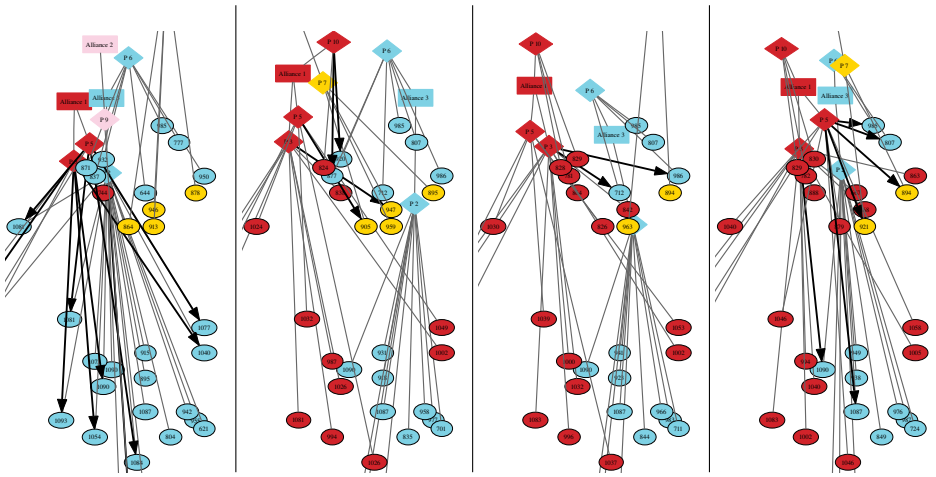


Figure 4.10: Travian game dynamics visualized as changes in the game graph (for $t = 1, 2, 3, 4, 5$). Bold arrows indicate conquest attacks by a player on a particular city.

dynamics, consider the sequence of game graphs shown in Figure 4.10. Here, three players from the red alliance launch a concerted attack against territory currently held by the blue and yellow alliances, and partially conquer it.

Data Collection and Preprocessing The data used in the experiments was collected from a "live" Travian server with approximately 25.000 active players. Over a period of three months (December 2007, January 2008, February 2008), high-level data about the current state of the game world was collected once every 24 hours. This included information about all cities, players, and the alliance structure in the game. For cities, their size and position on the map are available; for players, the list of cities they own; and for alliances the list of players currently affiliated with that alliance.

The game data was represented using predicates $city(C, X, Y, S, P)$ (city C of size S at coordinates X, Y held by player P), $allied(P, A)$ (player P is a member of alliance A), $conq(P, C)$ (indicating a conquest attack of player P on city C) and $alliance\_change(P, A)$ (player P changes affiliation to alliance A). A predicate $distance(C_1, C_2, D)$ with $D \in \{near, medium, far\}$ computing the (discretized) distance between cities was defined in the background knowledge. Sequences consist of between 29 and 31 such state descriptions.

Classification Experiments For the classification setting, we consider the problem of identifying so-called *meta-alliances* in Travian, which was recently introduced by Karwath et al. [2008]. A meta-alliance is a group of alliances that closely cooperate, thereby allowing large groups of players to work together. We manually identified meta-alliances in the collected game data based on the alliance names (a small free-text field). For instance, it is easy to recognize that the alliances *'.~A~.'*, *'.=A=.'*, and *'.-A-.'* are different wings of the same meta-alliance.

From all available game data 30 sequences of local game world states were extracted. Each sequence tracks a small set of players from three different alliances, two of which belong to the same meta-alliance this is indicated by a fact $meta\_alliance(a1, a2)$. On average, sequences consist of 25.8 interpretations, every interpretation contains 16.4 cities and 10.6 players, and there are 17.6 conquest events per sequence. The 30 extracted sequences constitute positive examples. A further 60 negative examples were obtained by giving the wrong meta-alliance information (i.e., $meta\_alliance(a1, a3)$ or $meta\_alliance(a2, a3)$).

We hand-coded a simple CPT-theory that encodes a few basic features that one would assume to be useful in such a task, such as whether two players in different alliances $a1$ and $a2$ attack each other (indicating $\neg meta\_alliance(a1, a2)$), or jointly attack a player from a third alliance (indicating $meta\_alliance(a1, a2)$). As an example, consider the following rule:

$$conq(C, P1) : 0.0061 \vee nil : 0.9939 \quad \longleftarrow$$
$$city(C, \_, \_, P2), player(P2, \_, A1),$$
$$player(P1, \_, A2), \neg meta\_alliance(A1, A2).$$

which states that the player $P1$ attacks a city C of a player $P2$ who is not his alliance partner.

Such a CPT-theory can be used for classification as follows. Given a set of training sequences $\mathcal{D}$, we first split this set into positive sequences $\mathcal{D}_+$ and negative sequences $\mathcal{D}_-$. We then learn the parameters of two CPT-theories $\mathcal{T}_+$ and $\mathcal{T}_-$ on the sets $\mathcal{D}_+$ and $\mathcal{D}_-$ according to maximum likelihood using the EM algorithm presented in Section 4.2.4. Note that $\mathcal{T}_+$ and $\mathcal{T}_-$ both employ the simple rule set outlined above, and only differ in their parameter values. Given a new test sequence S, we then evaluate the likelihood of S under the positive and negative models, $P(S \mid \mathcal{T}_+)$ and $P(S \mid \mathcal{T}_-)$, and predict the class for which this likelihood is higher.

To evaluate the accuracy of CPT-L in the meta alliance classification task, we performed a 10-fold cross-validation, using the same folds as used in [Karwath et al., 2008]. Figure 4.11 compares the results obtained for CPT-L with those of the BoostedReal system. BoostedReal is a state-of-the-art system for classification

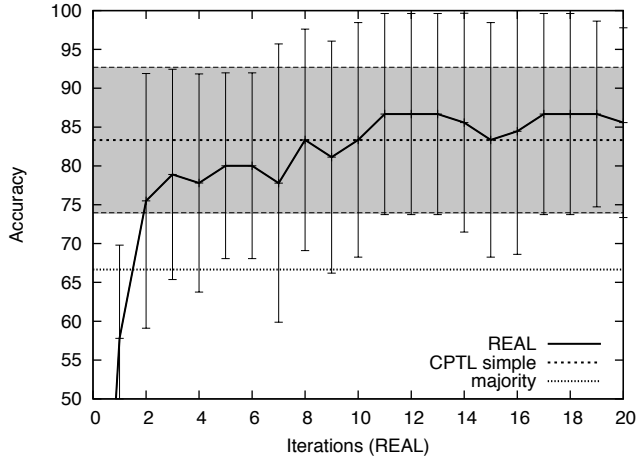


Figure 4.11: Classification accuracy for the BoostedReal system (see [Karwath et al., 2008]) and CPT-L for the meta-alliance problem in the massively multiplayer online game domain. For BoostedReal, accuracy is a function of the boosting iteration (shown on the x-axis). For CPT-L, standard deviation over the cross-validation folds is indicated by the shaded area. Classification accuracy of the majority-class predictor is also shown.

of (relational) sequences by alignment, which uses a discriminative approach based on boosting the reward model used in the alignment algorithm [Karwath et al., 2008]. Note that BoostedReal, in contrast to CPT-L, is not a generative model for sequences of interpretations, but rather a discriminative approach specifically tailored to classification problems. It is also significantly more complex, and the resulting models are harder to interpret, as the boosted reward function is represented as an ensemble of relational regression trees. Theoretically these trees can be collapsed into a single tree, but at the expanse of an exponential growth. Figure 4.11 shows that CPT-L, at 82.22% with standard deviation of 9.37, achieves a slightly lower accuracy than the best observed result for BoostedReal, although the difference is not significant. Overall, we can conclude from this experiment that even with the simple rule set used, CPT-L is able to learn a model that captures useful information about the positive and negative class, and achieves similar accuracies as other state-of-the-art sequence classification schemes. Learning a single model in this domain takes under 2 minutes, using the lifted inference technique described in Section 4.2.3.

We also tried to model this classification problem using discriminative Markov Logic Networks, in order to better understand the trade-offs between more general and simpler SRL approaches. However, with similar features as used in the CPT-L

rules described above, we have not been able to obtain classification accuracies higher than majority class. The time needed for building a model in Markov Logic is approximate 2 hours, which is about two orders of magnitude higher than for our approach.

Prediction Experiments We now consider the problem of predicting player actions within Travian, testing the prediction algorithm presented in Section 4.2.5. From all available data, we again extracted 30 sequences of local game world states. Each sequence involves a subset of 10 players, which are tracked over a period of one month (10 sequences each for December, January and February). Player sets are chosen such that there are no interactions between players in different sets, but a high number of interactions between players within one set. Cities that did not take part in any conquest event were removed from the data, leaving approximately 30–40 cities under consideration for every player subset.

We defined a world model in CPT-L that expresses the probability for player actions such as conquests of cities and changes in alliances affiliation, and updates the world state accordingly. Player actions in Travian—although strongly stochastic—are typically explainable from the social context of the game: different players from the same alliance jointly attack a certain territory on the map, there are retaliation attacks at the alliance level, or players leave alliances that have lost many cities in a short period of time. From a causal perspective, actions are thus triggered by certain (relational) patterns that hold in the game graph, which take into account a player's alliance affiliation together with the actions carried out by other alliance members. Such patterns can be naturally expressed in CPT-L as bodies of rules which trigger actions encoded in the head of the rule. We again manually defined a number of simple rules capturing such typical game patterns. As an example, consider the rules

$$\begin{aligned} conq(P,C) : 0.039 \vee nil : 0.961 \quad &\longleftarrow \quad conq(P,C'), city(C',\_,\_,\_,P'), \\ &\qquad\quad city(C,\_,\_,\_,P') \\ conq(P,C) : 0.011 \vee nil : 0.989 \quad &\longleftarrow \quad city(C,\_,\_,\_,P''), \\ &\qquad\quad allied(P,A), allied(P',A), \\ &\qquad\quad conq(P',C'), city(C',\_,\_,\_,P'') \end{aligned}$$

The first rule encodes that a player is likely to conquer a city of a player he or she already attacked in the previous time step. The second rule generalizes this pattern: a player P is likely to attack a city C of player P'' if an allied player has attacked P'' in the previous time step.

Moreover, the world state needs to be updated given the players' actions. After a conquest attack $conq(P,C)$, the city C changes ownership to player P in the next time step. If several players execute conquest attacks against the same city in one time step, one of them is chosen as the new owner of the city with uniform probability (note that such simultaneous conquest attacks would not be observed

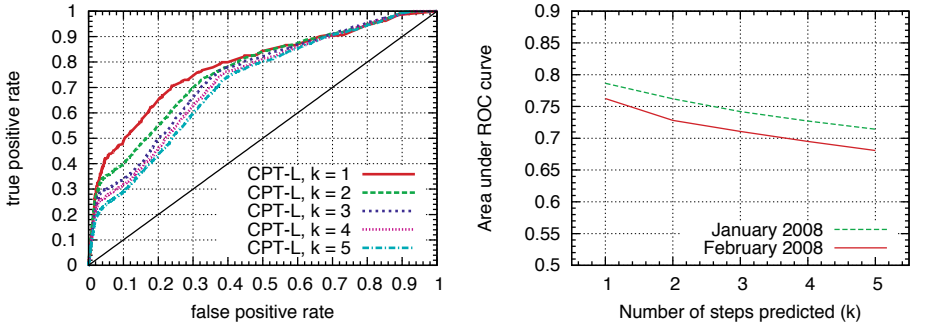


Figure 4.12: Left figure: ROC curve for predicting that a city C will be conquered by a player P within the next k time steps, for $k \in \{1, 2, 3, 4, 5\}$. The model was trained on 10 sequences of local game state descriptions from December 2007, and tested on 10 sequences from January 2008. Right figure: AUC as a function of the number k of future time steps considered in the same experiment. Additionally, AUC as a function of k is shown for 10 test sequences from February 2008.

in the training data, as only one snapshot of the world is taken every 24 hours). Similarly, an *alliance_change*(P, A) event changes the alliance affiliation of player P to alliance A in the next time step.

We now consider the task of predicting the "conquest" action *conq*(P, C) based on a learned model of world dynamics. The collected sequences of game states were split into one training set (sequences collected in December 2007) and two test sets (sequences collected in January 2008 and sequences collected in February 2008). Maximum-likelihood parameters of a hand-crafted CPT-theory $\mathcal{T}$ as described above were learned on the training set using EM. Afterwards, the learned model was used to predict the player action *conq*(P, C) on the test data in the following way. Let S denote a test sequence with states $I_0, \ldots, I_T$. For every $t_0 \in \{0, \ldots, T-1\}$, and every player p and city c occurring in S, the learned model is used to compute the probability that the conquest event *conq*(p, c) will be observed in the next world state, $P(I_{t_0+1} \models \mathit{conq}(p, c) \mid \mathcal{T}, I_0, \ldots, I_{t_0})$. This probability is obtained from the sampling-based prediction algorithm described in Section 4.2.5. The prediction is compared to the known ground truth (whether the conquest event occurred at that time in the game or not). Instead of predicting whether the player action will be taken in the next step, we can also predict whether it will be taken within the next k steps, by computing

$$P(I_{t_0+1} \models \mathit{conq}(p, c) \vee \ldots \vee I_{t_0+k} \models \mathit{conq}(p, c) \mid \mathcal{T}, I_0, \ldots, I_{t_0}).$$

This quantity is also easily obtained from the prediction algorithm described in Section 4.2.5.

Figure 4.12, left, shows ROC curves for this experiment with different values

$k \in \{1, 2, 3, 4, 5\}$, evaluated on the first test set (January 2008). Figure 4.12, right, shows the corresponding AUC values as a function of k for both test sets. The achieved area under the ROC curve is substantially above 0.5 (random performance), indicating that the learned CPT-theory $\mathcal{T}$ indeed captures some characteristics of player behavior and obtains a reasonable ranking of player/city pairs (p/c) according to the probability that p will conquer c. Moreover, the model is able to predict conquest actions several steps in the future, although AUC is slightly lower for larger k. This indicates that uncertainty associated with predictions accumulates over time. Finally, predictions for the first test set (January 2008) are slightly more accurate than for the second test set (February 2008). This is not surprising as the model has been trained from sequences collected in December 2007, and indicates a slight change in game dynamics over time. In summary, we conclude that player actions in Travian are indeed to some degree predictable from the social context of the game, and CPT-L is able to learn such patterns from the data. The computational complexity of learning in this task will be analyzed in detail in the next section.

Scaling Experiments We now analyze the scaling behavior of the proposed algorithms in detail, and compare the basic inference algorithm presented in Section 4.2.2 to the lifted inference algorithm presented in Section 4.2.3. To this end, we again consider the prediction setting discussed in the last section, and vary the number of players and cities that are present in any given game state. We used data containing up to 50 players, which together controlled up to 269 cities. As before, 30 sequences of such game states were extracted from the data. To evaluate computational complexity, a model was trained on all sequences, using the same rule set as was used for the prediction task.

To illustrate the complexity of the resulting problem, one can approximate the size of the ground network that would have been obtained had we grounded the model to a Bayesian or Markov Network as is typically done for exact inference in SRL approaches such as CP-logic or Markov Logic Networks. In such a network, nodes correspond to all groundings of predicates using available domain constants. Note, that in general, only the part of this network that is relevant for a particular query needs to be constructed. However, in our scenario all ground facts involving constants that appear in a training sequence are relevant when learning from that sequence or computing its likelihood. Furthermore, in the dynamic setting considered here, the network has to be unrolled over time, essentially duplicating the nodes for every time step in the observation sequence. For the largest domain we have considered (involving 50 players and 269 cities), the size of the ground network is approximately 800.000 nodes, indicating that exact inference and learning in this network would be computationally expensive.

Figure 4.13 shows the time needed to perform inference in CPT-L in the outlined domain as a function of the size of the (hypothetical) ground network, for up to 20

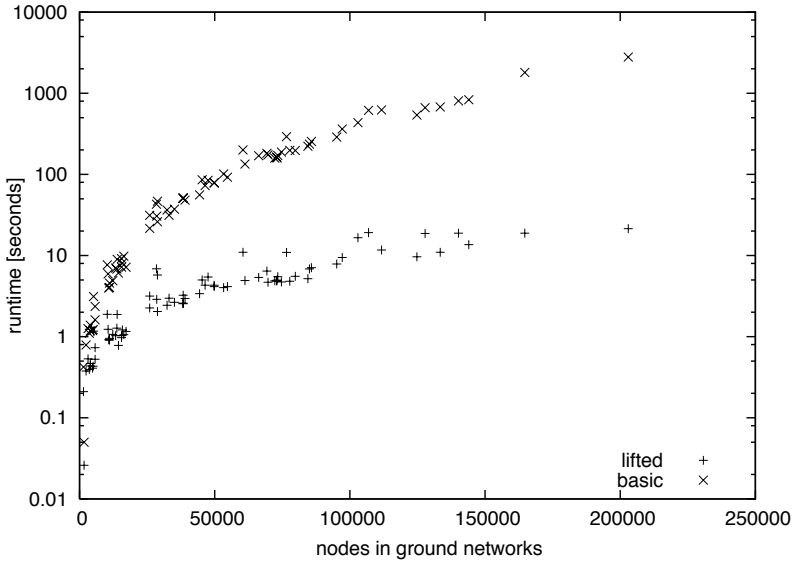


Figure 4.13: Time for performing inference (in the Expectation Step of the EM algorithm) for the Travian prediction task as a function of the domain size. The y-axis shows run time in seconds. The x-axis shows the number of nodes in the Bayesian network that would result from the grounding of the CPT-theory in this domain.

players. Timing results are given for both the basic inference algorithm presented in Section 4.2.2 and the lifted inference algorithm presented in Section 4.2.3. It can be observed that the lifted inference algorithm has significantly better scaling behavior, and achieves a speed-up of about a factor of 50 compared to the basic inference algorithm in large domains. For datasets containing more than 20 players, the standard inference algorithm could not be run anymore. However, we ran the lifted inference algorithm for datasets with up to 50 players, resulting in the (hypothetical) ground networks of approximately 800.000 nodes mentioned above. In this setting, lifted inference could still be performed in about 2 seconds.

Overall, these experiments show that the simple lifted inference algorithm yields a substantial speed-up compared to the basic inference algorithm. Note that the inference we perform is exact, and computational efficiency is achieved by exploiting the relative simplicity of our model and learning setting. This is in contrast to other approaches that try to overcome the excessive size of ground networks by performing approximate inference, as, for example, in Markov Logic Networks [Richardson and Domingos, 2006].

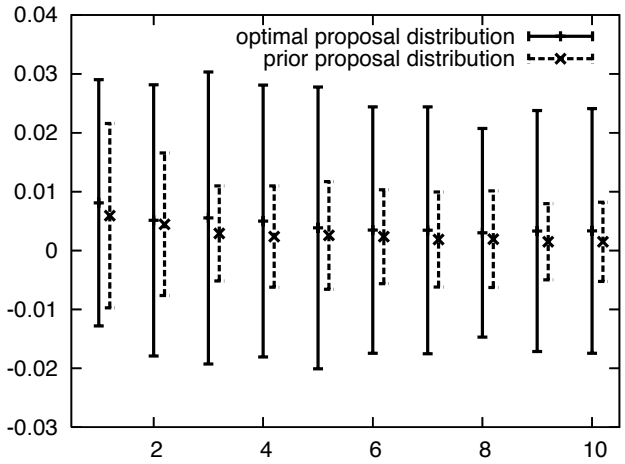


Figure 4.14: Effective number of particles divided by run time in dependence on sequence length.

4.3.4 Experiments in The Activity Recognition Domain.

We used the activity recognition domain to analyze the filtering algorithm. We were interested in comparing the results based on the optimal proposal distribution to the results based on the transition prior. In domains with deterministic dependencies between the hidden state and the observations the optimal proposal can be expected to outperform the transitions prior.

World Model The world model of the activity recognition domain contains objects from a typical office environment, for instance, books, a keyboard, pens, paper and so on. A person is observed while they perform one of several possible activities, e.g., writing, using a computer, doing nothing, making a phone call and so on. The change of activity is modeled by the rule

$$activity(working) : 0.01 \vee \ldots \vee activity(drinking) : 0.01 \vee$$
$$activity(X) : 0.83 \vee activity(idle) : 0.1 \longleftarrow activity(X)$$

This rule states that it is most likely that the current activity will be continued.

The observations provide cues which hint at the activity performed. One example is the pose of the person, which can indicate an activity, or just be a random pose.

$$pose(X) : 0.2 \vee pose(writing) : 0.1 \vee \ldots \vee pose(idle) : 0.1 \longleftarrow activity(X).$$

Other possible cues are due to the objects observed in the scene. We provide a relation, $useful/2$, in the background knowledge, which specifies that an object is useful for an activity, e.g., $useful(apple, typing)$ indicates that a computer is useful for typing. These relations are used by the rules

$$present(X) : 0.7 \vee nil : 0.3 \longleftarrow activity(A), useful(A, X).$$

$$present(X) : 0.1 \vee nil : 0.9 \longleftarrow useful(\_, X).$$

The first rule states that an object useful for the activity is observed with probability 0.7. The second rules states that all object are observed with at least probability 0.1

Note that the observations are such that no observation indicates an activity certainly or rules it out completely.

Results in the Activity Recognition Domain As the observation distribution has nowhere zero mass the transition-prior is expected to outperform the proposal distribution.

For the experiments we sampled 5 sequences of length 10. Afterwards we ran the particle filter algorithm with the optimal proposal distribution and the transition prior using 100 particles. For the optimal prior each run took less than a minute on a MacBook Pro 2.16Ghz. The transition prior was approximately 5 times faster. In figure 4.14 the effective number of particles (cf. Algorithm 3) divided by the run time in ms is plotted. The horizontal axis is the sequence length. Even though not significant, the optimal proposal distribution performed on average better. In toy example with spiked observation distribution the transition prior typically lost all particles in a few steps.

This result confirms that the invested run time pays off in terms of improved sample quality.

4.4 Related Work

There are relatively few existing approaches that can probabilistically model sequences of relational state descriptions. CPT-L can be positioned with respect to them as follows.

First, statistical relational learning systems such as Markov Logic [Richardson and Domingos, 2006], CP-logic [Vennekens et al., 2006], Probabilistic Relational Models [Getoor et al., 2001] or Bayesian Logic Programs [Kersting and De Raedt, 2007] can be used in this setting by adding an extra time argument to predicates (then called *fluents*). However, inference and learning in these systems is

computationally expensive: they support very general models including hidden states, and are not optimized for sequential data. A second class of techniques, for instance [Zettlemoyer et al., 2005], uses transition models based on (stochastic) STRIPS rules. This somewhat limits the transitions that can be expressed, as only one rule "fires" at every point in time, and it is difficult to model several processes that change the state of the world concurrently (such as an agent's actions and naturally occurring world changes). Related to this, is the probabilistic extension of PPDDL [Younes and Littman, 2004] that has been developed for the ICAPS planning competition and that form a generalization of STRIPS. From a representational perspective, PPDDL is equivalent to Dynamic Bayesian nets as actions in PPDDL are restricted to finite domains. PPDDL also employs frame axioms. However, we think that writing PPDDL is more difficult than CPT-L because the user is supposed to ensure that the theory is consistent and, hence, that consistency is not enforced by the language. This significantly complicates structure learning for PPDDL models. Nevertheless it is very well possible that the algorithms presented in this work can be adapted to PPDDL (see Section 8.1, page 133).

Another related formalism is that of Logical MDPs [Kersting and Raedt, 2004], which specifically targets Markov Decision Processes and thus takes into account rewards. The action rules employed in LoMDPs are somewhat similar to CPT-L rules, but they require that the bodies of the action rules are mutually exclusive (which is achieved by imposing an order on the rules). CP-logic, and therefore also CPT-L, neither imposes orders on rules, nor does it require that only one clause triggers at the same time, which makes it more natural to model stochastic relational processes.

Another approach designed to model sequences of relational state descriptions are relational simple-transition models [Fern, 2005]. A related approach employs dynamic Markov Logic to represent stochastic relational processes [Biswas et al., 2007]. Inference is carried out in a ground dynamic Bayesian network constructed from the MLN. In contrast to CPT-L, these two approaches focus on domains where the process generating the data is hidden, and the hidden states have to be inferred from observations. This is a significantly harder problem than the fully observable setting discussed in this paper, and therefore typically only approximate inference is possible [Fern, 2005]. However, we feel that also the easier problem where everything is observable is worthy of investigation in its own right. A better understanding of this problem should also provide new insights into the more complex one.

We also presented an extension of CPT-L to deal with hidden variables, which demonstrates how to apply these insights in a limited setting namely filtering. The algorithm for performing filtering is based on a Monte Carlo method.

4.5 Conclusions

We have introduced CPT-L, a probabilistic model for sequences of relational state descriptions. In contrast to other approaches that could be used as a model for such sequences, CPT-L focuses on computational efficiency rather than expressivity. We have specifically discussed how to perform efficient inference and parameter learning in CPT-L by a partially lifted inference algorithm. The algorithm aggregates all groundings of rules where the chosen head element is logically entailed into a joint factor during probabilistic inference, thereby significantly reducing the size of the resulting inference problem. We have also extended earlier work on CPT-L by relaxing the Markov assumption on the underlying stochastic process, and using more flexible rules where rule heads consist of a disjunction of conjunctions.

There are two main directions for future work. The first direction is structural optimization, that is, learning entire rule sets from data as opposed to only learning parameters for a given rule set. Experiments in this direction are promising but preliminary. A second interesting direction for future work is to extend the model towards a setting where data is only partially observed. We have started studying an extension of CPT-L in which a subset of domain predicates is hidden, while other predicates have to be fully observable. The resulting hidden state inference problem is computationally challenging, thus we use approximate inference techniques. Some initial encouraging results were achieved in this setting using particle filters. Finally, we are interested in applying the presented techniques in other challenging application domains.

Learning ProbLog Programs From Interpretations

5

This chapter shows how to learn the parameters of ProbLog programs if evidence is given by means of interpretations. The algorithm is based on the expectation maximization scheme. It is closely related to the two-step procedure for estimating the probabilities of the selections in CPT-L. In LFI-ProbLog, the first step generates a propositional logical formula for each training example. The more compact BDD representations of these formulas are used in in the second step to calculate the expected counts.

STATISTICAL relational learning [Getoor and Taskar, 2007] and probabilistic logic learning [De Raedt et al., 2008; De Raedt, 2008] have contributed various representations and learning schemes. Popular approaches include BLPs [Kersting and De Raedt, 2007], ICL [Poole, 2008], Markov Logic [Richardson and Domingos, 2006], PRISM [Sato and Kameya, 2001], PRMs [Getoor et al., 2001], and ProbLog [De Raedt et al., 2007; Gutmann et al., 2008]. These approaches differ not only in the underlying representations but also in the learning settings they employ.

This chapter builds on [Gutmann et al., 2011a], [Gutmann et al., 2010]

For learning knowledge-based model construction approaches (KBMC), such as Markov Logic, PRMs, and BLPs, one typically uses relational state descriptions as training examples. This setting is also known as *learning from interpretations*. For training probabilistic programming languages one typically uses *learning from entailment* [De Raedt and Kersting, 2004; De Raedt, 2008]. In particular, probabilistic logic programming (PLP) typically, based on Sato's distribution semantics [Sato, 1995] such as PRISM and ProbLog use training examples in the form of labeled facts. The labels are either the truth values of these facts or target probabilities.

In the learning from entailment setting, one usually starts from observations for a *single target* predicate. In the learning from interpretations setting, however, the observations specify the value for some of the random variables in a state-description. Probabilistic grammars and graphical models are illustrative examples for each setting. Probabilistic grammars are trained on training examples in the form of sentences. Each training example states whether a particular sentence was derived or not, but it does not explain *how* it was derived. In contrast, Bayesian networks are typically trained on partial or complete state descriptions, which specify the value for some random variables in the network. This also implies that training examples for Bayesian networks can contain much more information. These differences in learning settings also explain why the KBMC and PLP approaches have been applied to different kinds of data sets and applications. Entity resolution and link prediction are examples of domains where KBMC has been successfully applied. We aim at bridging the gap between these two types of approaches to learning. We study how the parameters of ProbLog programs can be learned from partial interpretations. The **key contribution** of this chapter is a novel algorithm, called LFI-ProbLog, that is used for learning ProbLog programs from partial interpretations. The algorithm generalizes the algorithm for learning parameters of CPT-L models (Section 4.2.4) in two ways. First the generation of the propositional formula and evaluation of the BDDs allow unobserved facts. Second instead of hard-coding independence assumptions like for CPT-L, the algorithm automatically detects and exploits conditional independence. We thoroughly evaluated the algorithm on various standard benchmark problems.

This chapter is organized as follows: Section 5.1 formalizes the problem of learning the parameters of ProbLog programs from interpretations. Section 5.2 introduces LFI-ProbLog for finding parameters which maximize the likelihood. We report on experimental results in Section 5.3. Before concluding, we discuss related work in Section 5.4.

5.1 Learning from Interpretations

Learning from (possibly partial) interpretations is a common setting in statistical relational learning that has not yet been studied in its full generality for probabilistic programming languages. A ProbLog program specifies a distribution over interpretations as outlined in Section 3.1 and Section 3.2. It basically follows from the fact that each total choice generates a unique least Herbrand interpretation. Therefore learning from interpretations is a natural learning setting in ProbLog. Often not all facts are observable, but only a subset of facts is known to be "*true*" or "*false*". A **partial interpretation** I specifies for some (but not all) atoms the truth value. We represent partial interpretations as $I = (I^+, I^-)$ where I^+ contains all "*true*" atoms and I^- all "*false*" atoms. A partial interpretation needs to be consistent that is I^+ and I^- need to be mutually exclusive. The probability of a partial interpretation is the sum of the probabilities of all possible worlds consistent with the known atoms. Expressed in terms of the distribution semantics (cf. Section 3.1, page 34), this corresponds to $P_{\mathcal{T}}(I)$.

Alternatively, the probability of the partial interpretation I can be calculated as the success probability of the query $\Phi(I) = \Phi((I^+, I^-)) := (\bigwedge_{a_j \in I^+} a_j) \wedge (\bigwedge_{a_j \in I^-} \neg a_j)$. Considering Example 3.3, the probability of the following partial interpretation

$$I^+ = \{\texttt{person(mary)}, \texttt{person(john)}, \texttt{burglary}, \texttt{alarm},$$
$$\texttt{calls(john)}, \texttt{al(john)}\}$$
$$I^- = \{\texttt{calls(mary)}, \texttt{al(mary)}\}$$

is $P(I) := P_{\mathcal{T}}([\![\Phi(I^+, I^-)]\!]) = \big(0.1 \times 0.7\big) \times \big((1 - 0.7)\big) \times \big((0.2 + (1 - 0.2))\big)$.

In a generative setting like the one defined by the extension process, one is typically interested in the maximum likelihood parameters given the training data. This task can be formalized as follows.

> **Max-Likelihood Parameter Estimation: Given** a ProbLog program $\mathcal{T}(\mathbf{p})$ containing the probabilistic facts $\mathcal{F}_l$ with *unknown* parameters $\mathbf{p} = \langle p_1, ..., p_N \rangle$ and background knowledge $\mathcal{BK}$, and a set of (possibly partial) interpretations $\mathcal{D} = \{I_1, \ldots, I_M\}$ (the training examples). **Find** maximum likelihood probabilities $\widehat{\mathbf{p}} = \langle \widehat{p_1}, \ldots, \widehat{p_N} \rangle$ such that
>
> $$\widehat{\mathbf{p}} = \arg\max_{\mathbf{p}} P(\mathcal{D}|\mathbf{p}) = \arg\max_{\mathbf{p}} \prod_{m=1}^{M} P_{\mathcal{T}(\mathbf{p})}(I_m)$$
>
> Thus, we are given a ProbLog program and a set of partial interpretations and the goal is to find the maximum likelihood parameters.

One has to consider two cases when computing $\widehat{\mathbf{p}}$. For complete interpretations where everything is observable, one can compute $\widehat{\mathbf{p}}$ by counting (cf. Sect. 5.1.1). In the more complex case of partial interpretations, one has to use an approach that is capable of handling partial observability (cf. Sect. 5.1.2).

5.1.1 Full Observability

It is clear that in the fully-observable case the maximum likelihood estimators $\widehat{p_n}$ for the probabilistic facts $p_n :: f_n$ can be obtained by counting the number of *true* ground instances in every interpretation, that is,

$$\widehat{p_n} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{k=1}^{K_n^m} \delta_{n,k}^m \quad \text{, where} \quad \delta_{n,k}^m := \begin{cases} 1 & \text{if } f_n\theta_{n,k}^m \in I_m \\ 0 & \text{else} \end{cases} \tag{5.1}$$

and $\theta_{n,k}^m$ is the k-th possible grounding substitution for the fact f_n in the interpretation I_m and K_n^m is the number of such substitutions. The sum is normalized by $Z_n = \sum_{m=1}^{M} K_n^m$, the total number of ground instances of the fact f_n in all training examples. If Z_n is zero, i.e., no ground instance of f_n is used, $\widehat{p_n}$ is undefined and one must not update p_n.

Before moving on to the partially observable case, let us consider the issue of determining the possible substitutions $\theta_{n,k}^m$ for a fact $p_n :: f_n$ and an interpretation I_m. To resolve this, we assume that the facts f_n are typed, and that each interpretation I_m contains an explicit definition of the different types in the form of a (fully observable) unary predicate.[1] In the alarm example 3.3, the predicate `person/1` can be regarded as the type of the (first) argument of `al(X)` and `calls(X)`. This predicate can differ between interpretations. One person, i.e., can have `john` and `mary` as neighbors, another one `ann`, `bob` and `eve`.

5.1.2 Partial Observability

In many applications the training examples are partially observed. In the alarm example, we may receive a phone call but we may not know whether an earthquake has in fact occurred. In the partial observable case – similar to Bayesian networks – a closed-form solution of the maximum likelihood parameters is infeasible. Instead, one has to replace in Equation (5.1) the term $\delta_{n,k}^m$ by $\mathbb{E}_{\mathcal{T}}[\delta_{n,k}^m | I_m]$, i.e., the conditional expectation given the partial interpretation under the current model $\mathcal{T}$,

$$p_n^{(new)} = \frac{1}{Z_n} \sum_{m=1}^{M} \sum_{k=1}^{K_n^m} \mathbb{E}_{\mathcal{T}}[\delta_{n,k}^m | I_m] \ . \tag{5.2}$$

[1] This assumption can be relaxed if the types are computable from the Problog program and the current interpretation.

By repeatedly updating the parameters they will converge towards a local maximum in the likelihood space. As in the fully observable case, the domains are assumed to be given. Before describing the EM algorithm for finding $\widehat{p_n}$, we illustrate one of its crucial properties using the alarm example. Assume that our partial interpretation is $I^+ = \{\texttt{person(mary)}, \texttt{person(john)}, \texttt{alarm}\}$ and $I^- = \varnothing$. It is clear that for calculating the marginal probabilities of all probabilistic facts – these are the expected counts – only the atoms in {`burglary`, `earthquake`, `al(john)`, `al(mary)`, `alarm`}$\cup I$ are relevant. This is due to the fact that the remaining atoms {`calls(john)`, `calls(mary)`} cannot be used in any proof for the facts observed in the interpretations. Therefore, they do not influence the probability of the partial interpretation.[2] This motivates the following definition of the dependency set.

Let $\mathcal{T} = \mathcal{F}_l \cup \mathcal{BK}$ be a ProbLog theory, $\mathcal{F}$ the facts defined by $\mathcal{F}_l$ and x a ground atom then the **dependency set of** x is defined as:

$$\text{dep}_{\mathcal{T}}(x) = \{f \text{ ground fact} \mid \text{a ground SLD-proof in } \mathcal{T} \text{ for } x \text{ contains } f\} \ . \quad (5.3)$$

Thus, $\text{dep}_{\mathcal{T}}(x)$ contains all ground atoms that appear in any possible proof of the atom x and it is finite due to the finite support condition

The dependency set of a fact can be generalized to partial interpretations: Let $\mathcal{T} = \mathcal{F}_l \cup \mathcal{BK}$ be a ProbLog theory and $I = (I^+, I^-)$ a partial interpretation then the *dependency set of the partial interpretation* I is defined as

$$\text{dep}_{\mathcal{T}}(I) = \bigcup\nolimits_{x \in (I^+ \cup I^-)} \text{dep}_{\mathcal{T}}(x) \ , \quad (5.4)$$

This allows us to restrict the probability calculation to the finite set of dependent atoms only. Before doing so, we first need the notion of a restricted ProbLog theory. Let $\mathcal{T} = \mathcal{F}_l \cup \mathcal{BK}$ be a ProbLog theory and $I = (I^+, I^-)$ a partial interpretation. Then we define $\mathcal{T}^r(I) = \mathcal{F}_l^r(I) \cup \mathcal{BK}^r(I)$, the *interpretation-restricted* ProbLog theory, as follows.

$$\mathcal{F}_l^r(I) = \{p_i :: f_i\theta \mid p_i :: f_i \in \mathcal{F}_l \text{ and } f_i\theta \in \text{dep}_{\mathcal{T}}(I)\} \quad (5.5)$$

$$\mathcal{BK}^r(I) = \{c\theta \ \mid \ c = h :\!- b_1, \ldots, b_n \in \mathcal{BK} \text{ and } h\theta \in \text{dep}_{\mathcal{T}}(I) \quad (5.6)$$
$$\text{and } b_i\theta \in \text{dep}_{\mathcal{T}}(I)\}$$

It is obtained by computing all ground instances of clauses in $\mathcal{BK}$ in which all atoms appear in $\text{dep}_{\mathcal{T}}(I)$. For instance, for the partial interpretation

$$I = (\{\texttt{burglary}, \texttt{alarm}\}, \varnothing)$$

[2] Such atoms play a role similar to that of barren nodes in Bayesian networks [Jensen, 2001].

the restricted program is

$$\mathcal{BK}^r(I) = \{\texttt{alarm} :- \texttt{burglary},\ \texttt{alarm} :- \texttt{earthquake}\}$$

$$\mathcal{F}_l^r(I) = \{\texttt{burglary},\ \texttt{earthquake}\}.$$

The following theorem shows that the conditional probability of f_n given I calculated in the theory $\mathcal{T}$ is equivalent to the probability calculated in $\mathcal{T}^r(I)$. The probabilistic facts of $\mathcal{T}^r(I)$ are restricted to the atomic choices occurring in $\text{dep}_\mathcal{T}(I)$. The restricted theory $T^r(I)$ cannot be larger than $\mathcal{T}$. More importantly, $T^r(I)$ is always finite.[3] In many cases it will be much smaller, which allows for learning in domains where the original theory does not fit in memory.

Theorem 5.1 *For all ground probabilistic facts f_n and partial interpretations I_m*

$$\mathbb{E}_\mathcal{T}[\delta^m_{n,k}|I_m] = \begin{cases} \mathbb{E}_{\mathcal{T}^r(I_m)}[\delta^m_{n,k}|I_m] & \text{if } f_n \in \text{dep}_\mathcal{T}(I_m) \\ p_n & \text{otherwise} \end{cases}$$

where $\mathcal{T}^r(I_m)$ is the interpretation-restricted ProbLog theory of $\mathcal{T}$ and p_n is the probability of the fact f_n. We assume that the naming of $\delta^m_{n,k}$ in $\mathbb{E}_{\mathcal{T}^r(I_m)}$ is such that the association of the facts in $\mathcal{F}_l$ and their groundings in $\mathcal{F}_l^r(I_m)$ is preserved.

5.2 The LFI-ProbLog algorithm

We now develop the EM algorithm for tight ProbLog programs, which calculates the parameters $\widehat{\mathbf{p}}$ defined in Equation (5.2) by maximizing the likelihood. A ProbLog program is **tight** or **positive-order-consistent** if there is no (positive) cycle in the positive dependency graph. An h atom depends on an atom b, if b appears as a positive in a clause with head h. This is due to the use of Clark's[Nilsson and Małuszyński, 1990] completion, which preserves the Herbrand semantics only for tight programs [Fages, 1994; Erdem and Lifschitz, 2003].

The algorithm proceeds in three steps. First, it constructs a CNF weighted by probabilities. Second, it simplifies and tries to split the CNF. Finally it builds Binary Decision Diagram(s) (BDD) [Bryant, 1986] for every training example I_m (cf. Sect. 5.2.1). These can then used to compute the expected counts $\mathbb{E}[\delta^m_{n,k}|I_m]$ and update the weights (cf. Sect. 5.2.3) .

The BDD represents the conditions under which the partial interpretation will be generated by the ProbLog program and the variables in the formula are the ground atoms in $\text{dep}_\mathcal{T}(I_m)$.

[3] Due to the finite support property and that the evidence is a finite set of ground atoms.

1.) Calculate Dependencies:

$$\text{dep}_{\mathcal{T}}(\texttt{alarm}) = \{\texttt{alarm}, \texttt{earthquake}, \texttt{burglary}\}$$

$$\text{dep}_{\mathcal{T}}(\texttt{calls(john)}) = \{\texttt{burlary}, \texttt{earthquake}, \texttt{al(john)}, \texttt{alarm}\}$$

$$\texttt{person(john)}, \texttt{calls(john)}\}$$

2.) Restricted theory:

0.1 :: burglary. person(john).

0.2 :: earthquake. alarm :− burglary.

0.7 :: al(john). alarm :− earthquake.

calls(john) :− person(john), alarm, al(john).

3.) Clark's completion:

$$\texttt{person(john)} \leftrightarrow \texttt{true}$$

$$\texttt{alarm} \leftrightarrow (\texttt{burglary} \vee \texttt{earthquake})$$

$$\texttt{calls(john)} \leftrightarrow \texttt{person(john)} \wedge \texttt{alarm} \wedge \texttt{al(john)}$$

4.) Propagated evidence:
$(\texttt{burglary} \vee \texttt{earthquake}) \wedge \neg\texttt{al(john)}$

5.) Build and evaluate BDD

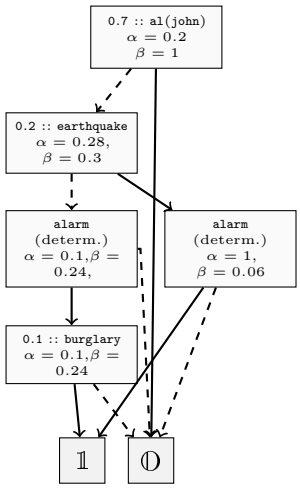


Figure 5.1: The different steps of the LFI-ProbLog algorithm for the training example $I^+ = \{\texttt{alarm}\}$, $I^- = \{\texttt{calls(john)}\}$. Normally the alarm node in the BDD is propagated away in Step 4, but is kept here for illustrative purposes. The nodes are labeled with their probability and the up- and downward probabilities.

5.2.1 Computing the BDD for an interpretation

The LFI-ProbLog algorithm consists of several steps, which generate the BDD that encodes a partial interpretation I. These steps are (cf. Fig 5.1):

1. Compute $\text{dep}_{\mathcal{T}(I)}$. This is the set of ground atoms on which the atoms with known truth value in the partial interpretation I depend. This is realized by applying the definition of $\text{dep}_{\mathcal{T}(I)}$ directly using a tabled meta-interpreter in Prolog. We use tabling, this is we store previously succeeded sub-goals to avoid recomputation.

2. Use $\text{dep}_{\mathcal{T}(I)}$ to compute $\mathcal{BK}^r(I)$, the background theory $\mathcal{BK}$ restricted to the interpretation I (cf. Equation (5.5) and Theorem 5.1).

3. Compute $clark(\mathcal{BK}^r(I))$, which denotes Clark's completion of $\mathcal{BK}^r(I)$; it is computed by replacing all clauses with the same head h :– $body_1$, ..., h :– $body_n$ by the corresponding formula $h \leftrightarrow body_1 \vee \ldots \vee body_n$. Clark's completion allows one to propagate values from the head to the bodies and vice versa. It states that the head is *true* if *and only if* at least one of its bodies is *true*. The completion captures the least Herbrand model semantics for tight programs.

4. Simplify $clark(\mathcal{BK}^r(I))$ by propagating known values for the atoms in I. This step eliminates ground atoms with known truth value in I. That is, we simply fill out their value in the theory $clark(\mathcal{BK}^r(I))$, and then we propagate these values until no further propagation is possible. This is akin to the first steps of the Davis-Putnam algorithm.

5. Construct the BDD_I, which compactly represents the Boolean formula consisting of the resulting set of clauses. This BDD_I is used by Algorithm 4 outlined in Section 5.2.3 to compute the expected counts.

In Step 4 of the algorithm, atoms f_n in the formula with known truth values v_n are replaced by the truth value and in turn removed from the BDD. This has to be taken into account both when calculating the probability of the interpretation and the expected counts of these variables. The probability of the partial interpretation I given the ProbLog program $\mathcal{T}(\mathbf{p})$ can be calculated as:

$$P_w(I|\mathcal{T}(\mathbf{p})) = P(BDD_I) \prod_{f_n \text{ known in } I} P(f_n = v_n) \tag{5.7}$$

where v_n is the value of f_n in I and $P(BDD_I)$ is the probability of the BDD as defined in the following subsection. The probability calculation is implemented using Equation (5.7). For the ease of notation, however, we shall act as if BDD_I included the variables corresponding to probabilistic facts with known truth value in I.

Furthermore, for computing the expected counts, we also need to consider the nodes and atoms that have been removed from the Boolean formula when the BDD has been computed in a compressed form. See for example in Fig. 5.1 (5) the probabilistic fact `burglary`. It only occurs on the left path to the $\mathbb{1}$-terminal, but it is with probability 0.1 also *true* on the right path. Therefore, we treat missing atoms at a particular level as if they were there and simply go to the next node no matter whether the missing atom has the value *true* or *false*.

5.2.2 Automated theory splitting

For large ground theories the naïvely constructed BDDs are too large to fit in memory. BDD tools use heuristics to find a variable order that minimizes the size of the BDD. The run-time of this step is exponential in the size of the input, which is prohibitive for parameter learning. We propose an algorithm that identifies independent parts of the grounded theory $clark(\mathcal{BK}^r(I))$ (the output of Step 4). The key observation is that the BDD for the Boolean formula $A \wedge B$ can be decomposed into two BDDs, one for BDD for A and one for B respectively, if A and B do not share a common variable. Since each variable is contained in at most one BDD, the expected counts of variables can be computed as the union of the expected count calculation on both BDDs. In order to use the automatic theory splitting, one has to replace Step 5 of the BDD construction (cf. Section 5.2.1) with the following algorithm. The idea is to identify sets of independent formulas in a theory by mapping the theory onto a graph as follows.

1. Add one node per clause in $clark(\mathcal{BK}^r(I))$.
2. Add an edge between two nodes if the corresponding clauses share an atom.
3. Identify the connected components in the resulting graph.
4. Build for each of the connected components one BDD representing the conjunction of the clauses in the component.

The resulting set of BDDs are used by the algorithm outlined in the next section to compute the expected counts.

5.2.3 Calculate Expected Counts

Similarly to the algorithm for estimating the parameters of a CPT-theory (Section 4.2.2, page 58), one can calculate the expected counts $\mathbb{E}[\delta^m_{n,k}|I_m]$ by a dynamic programming approach on the BDD. In both cases the training examples are represented as propositional formulas, which can be represented as a BDD. However

there are several important differences. The most important one is that there is no node/proposition representing the negative outcome. Hence the negative branch has to be multiplied with the probability of the negative outcome, this is akin to Equation (3.3). The second difference is that we extend the algorithm to handle deterministic nodes, which do not govern a probability distribution, but depend on the choices in the BDD. Lastly, a probabilistic fact might be missing on a path. This is due to the removal of nodes with isomorphic children.

To keep notation uncluttered, we will conceptually make two changes to the BDD. First nodes with isomorphic children are not removed. Ignoring nodes which do not occur on a path would result in an underestimate of the expected counts. Therefore, we assume that those nodes are contained in the BDD with isomorphic children. The second assumption we make is that all propagated values are added back to the BDD as $f_i \Leftrightarrow v_i$.

We use p_N as the probability that the node N will be left using the branch to the high-child and $1 - p_N$ otherwise. For a node N corresponding to a probabilistic fact f_i this probability is $p_N = p_i$ and $p_N = 1$ otherwise. We use the indicator function $\pi_N = 1$ to test whether a node N is deterministic. For every node N in the BDD we compute:

1. The *upward probability* $\alpha(N)$ represents the probability that the logical formula encoded by the sub-BDD rooted at N is *true*. For instance, in Fig. 5.1 (5), the upward probability of the leftmost node for `alarm` represents the probability that the formula `alarm` $\wedge$ `burglary` is *true*.

2. The *downward probability* $\beta(N)$ represents the probability of reaching the current node N on a random walk starting at the root, where at deterministic nodes both paths are followed in parallel. If all random walkers take the same decisions at the remaining nodes it is guaranteed that only one reaches the $\mathbb{1}$-terminal. This is due to the fact that the values of all deterministic nodes are fixed given the values for all probabilistic facts. For instance, in Fig. 5.1 (5), the downward probability of the left `alarm` node is equal to the probability of $\neg$`earthquake` $\wedge \neg$`al(john)`, which is $(1 - 0.2) \cdot (1 - 0.7)$.

The following invariants hold for the BDD:

$$\sum_{N \text{ node at level } n} \alpha(N) \cdot \beta(N) = P(BDD) \ ,$$

where the variable N ranges over all nodes occurring at a particular level n in the BDD. Each path from the root to the $\mathbb{1}$-terminal corresponds to an assignment of values to the variables that satisfies the Boolean formula underlying the BDD. The probability that such a path passes through the node N can be computed as $\alpha(N) \cdot \beta(N) \cdot (P(BDD))^{-1}$. The upward and downward probabilities are computed

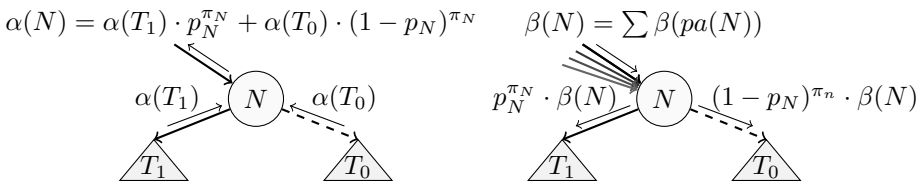


Figure 5.2: Propagation step of the upward probability (left) and for the downward probability (right). The indicator function π_N is 1 if N is a probabilistic node and 0 otherwise.

using the following formulae (cf. Fig. 5.2):

$$\alpha(\mathbb{0}) = 0 \qquad \alpha(\mathbb{1}) = 1 \qquad \beta(\mathit{Root}) = 1$$

$$\alpha(N) = \alpha(h(N)) \cdot p_N^{\pi_N} + \alpha(l(N)) \cdot (1 - p_N)^{\pi_N}$$

$$\beta(N) = \sum\nolimits_{N=h(M)} \beta(M) \cdot p_M^{\pi_M} + \sum\nolimits_{N=l(M)} \beta(M) \cdot (1 - p_M)^{\pi_M} \ ,$$

where π_N is 0 for nodes representing deterministic nodes and 1 otherwise. Due to the definition of α and β, the probability of the BDD is returned both at the root and at the $\mathbb{1}$-terminal, that is, $P(BDD) = \alpha(\mathit{Root}) = \beta(\mathbb{1})$. Given these values, one can compute the expected counts $\mathbb{E}[\delta_{n,k}^m | I_m]$ as

$$\mathbb{E}[\delta_{n,k}^m | I_m] = \sum\nolimits_{N \text{ represents } f_m} \frac{\beta(N) \cdot p_N \cdot \alpha(h(N))}{P(BDD)} \ .$$

One computes the downward probability α from the root to the leaves and the upward probability β from the leaves to the root. Intermediate results are stored and reused when nodes are revisited. Both parts are sketched in Algorithm 4.

5.3 Experimental evaluation of LFI-ProbLog

We implemented LFI-ProbLog in YAP Prolog [Santos Costa et al., 2011] and use CUDD [Somenzi, 2009] for the BDD operations. We used two datasets to evaluate LFI-ProbLog. The *WebKB* benchmark serves as test case to compare with state-of-the-art systems. The *Smokers* datatset is used to test the algorithm in terms of the learned model, that is, how close the parameters are to the original ones. The experiments were run on an Intel Core 2 Quad machine (2.83 GHz) with 8GB RAM.

Algorithm 4 Calculating the upward α and downward probabilityβ of the nodes in a BDD

```
function ALPHA(BDD node N)
    if ALPHA(N) already known then return ALPHA(N)
    if N is the 1-terminal then return 1
    if N is the 0-terminal then return 0
    let h and l be the high and low children of N
    if N represents a probabilistic fact then
        return p_N · ALPHA(h) + (1 − p_N) · ALPHA(l)
    else
        return ALPHA(h) + ALPHA(l)


function BETA(BDD node N)
    q priority queue, with sorting according to BDD-order
    enqueue(q, N)
    initialize BETA as array of 0's, with on entry per BDD node.
    while q not empty do
        N = dequeue(q)
        if N represents a probabilistic fact then
            BETA[h(N)]+ = BETA[N] · p_N
            BETA[l(N)]+ = BETA[N] · (1 − p_N)
        else
            BETA[h(N)]+ = BETA[N]
            BETA[l(N)]+ = BETA[N]
        enqueue(q, h(N)) if not yet in q
        enqueue(q, l(N)) if not yet in q
```

5.3.1 WebKB

The goal of this experiment is to answer the following questions:

Q1 *Is LFI-ProbLog competitive with existing state-of-the-art frameworks?*

Q2 *Is LFI-ProbLog insensitive to the initial probabilities?*

Q3 *Is the theory splitting algorithm capable of handling large data sets?*

In this experiment, we used the WebKB dataset [Craven and Slattery, 2001]. It contains four folds, each describing the link structure of pages from one of the following universities: Cornell, Texas, Washington, and Wisconsin. WebKB is a collective classification task, that is, one wants to predict the class of a page depending on the classes of the pages that link to it and depending on the words

being used in the text. To allow for an objective comparison with Markov Logic networks and the results of Domingos and Lowd [2009], we used their slightly altered version of WebKB. In their setting each page is assigned exactly one of the classes "course", "faculty", "other", "researchproject", "staff", or "student". Furthermore, the class "person", present in the original version, has been removed.

We use the following model that contains one non-ground probabilistic fact for each pair of CLASS and WORD. To account for the link structure, it contains one non-ground probabilistic fact for each pair of CLASS1 and CLASS2.

$$\mathrm{P}_{Word(\textsc{Class},\textsc{Word})} :: \texttt{pfWoCla}(\texttt{Page}, \textsc{Class}, \textsc{Word}).$$

$$\mathrm{P}_{Link(\textsc{Class1},\textsc{Class2})} :: \texttt{pfLiCla}(\texttt{Page1}, \texttt{Page2}, \textsc{Class1}, \textsc{Class2}).$$

If a ground instance $\texttt{pfWoClass}(\texttt{url}, \texttt{w}, \texttt{c})$ of the probabilistic fact `pfWoCla/3` is *true* a page containing word `w` will have the corresponding class `c`. The probabilistic fact `pfLiCla/4` represents the influence of the class label given that one page has a link to the other one. The probabilities of these facts are unknown and have to be learned by LFI-ProbLog. As there are 6 classes and 771 words, our model has $6 \times 771 + 6 \times 6 = 4662$ parameters. In order to combine the probabilistic facts and predict the class of a page we add the following background knowledge.

$$\texttt{cl}(\texttt{Pa}, \texttt{C}) :- \texttt{hasWord}(\texttt{Pa}, \texttt{Word}),\ \texttt{pfWoCla}(\texttt{Pa}, \texttt{Word}, \texttt{C}).$$

$$\texttt{cl}(\texttt{Pa}, \texttt{C}) :- \texttt{linksTo}(\texttt{Pa2}, \texttt{Pa}),\ \texttt{pfLiCla}(\texttt{Pa2}, \texttt{Pa}, \texttt{C2}, \texttt{C}),\ \texttt{cl}(\texttt{Pa2}, \texttt{C2}).$$

If the determinstic and probabilistic facts in the body of each clause are true the page gets assigned the corresponding class.

We performed a 4-fold cross validation, that is, we trained the model on three universities and then tested it on the fourth one. We repeated this for all four universities and averaged the results. We measured the area under the precision-recall curve (AUC-PR), the area under the ROC curve (AUC-ROC), the log likelihood, and the accuracy after each iteration of the EM algorithm. Our model does not express that each page has exactly one class. To account for this, we normalize the probabilities per page.

Figure 5.3 (left) shows the AUC-ROC plotted against the average training time. The initialization phase, that is running steps 1-4 of LFI-ProbLog, takes ≈ 330 seconds, and each iteration of the EM algorithm takes ≈ 62 seconds. We initialized the probabilities of the model randomly with values sampled from the uniform distribution between 0.1 and 0.9, which is shown as the graph for LFI-ProbLog [0.1-0.9]. After 10 iterations (≈ 800 s) the AUC-ROC is 0.950 ± 0.002, the AUC-PR is 0.828 ± 0.006, and the accuracy is 0.769 ± 0.010.

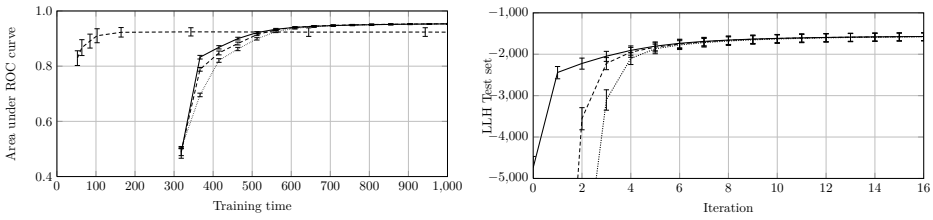


Figure 5.3: Area under the ROC curve against the learning time (left) and test set log likelihood for each iteration of the EM algorithm (right) for WebKB.

We compared LFI-ProbLog with Alchemy [Domingos and Lowd, 2009] and LeProbLog [Gutmann et al., 2008]. Alchemy is an implementation of Markov Logic networks. We use the model suggested by Domingos and Lowd [2009], which uses the same features as our model, and we train it according to their setup.[4] The learning curve for AUC-ROC is shown in Figure 5.3 (left). After 943 seconds Alchemy achieves an AUC-ROC of 0.923 ± 0.016, an AUC-PR of 0.788 ± 0.036, and an accuracy of 0.746 ± 0.032. LeProbLog is a regression-based parameter learning algorithm for ProbLog. The training data has to be provided in the form of queries annotated with the target probability. It is not possible to learn from interpretations. For WebKB, however, one can map one interpretation to several training examples $P(\texttt{class}(\text{URL},\textsc{Class})) = \text{P}$ per page where P is 1 if the class of URL is Class and else 0. This is possible, due to the existence of a *target predicate*. We used the standard settings of LeProblog[5] and limit the run-time to 24 hours. Within this limit, the algorithm performed 35 iteration of gradient descent. The final model obtained an AUC-PR of 0.419 ± 0.014, an AUC-ROC of 0.738 ± 0.014, and an accuracy of 0.396 ± 0.020. These results answer **Q1** positive.

We tested how sensitive LFI-ProbLog is for the initial fact probabilities by repeating the experiment with values sampled uniformly between 0.1 and 0.3 and sampled uniformly between 0.0001 and 0.0003 respectively. As the graphs in Figure 5.3 indicate, the convergence is initially slower and the initial log-likelihood values differ. This is due to the fact that the ground truth probabilities are small, and if the initial fact probabilities are small too, one obtains a better initial log-likelihood. All settings converge to the same results, in terms of AUC and log-likelihood. This suggests that LFI-ProbLog is insensitive to the start values (cf. **Q2**).

The BDD for the WebKB dataset are too large to fit in memory and the automatic variable reordering is unable to construct the BDD in a reasonable amount of time. We used two different approaches to resolve this. In the first approach, we manually split each training example, that is, the grounded theory together with the known

[4] Daniel Lowd provided us with the original scripts for the experiment setup. We report on the evaluation based on the rerun of the experiment.

[5] available at `http://dtai.cs.kuleuven.be/problog/`

class for each page, into several training examples. The results shown in Figure 5.3 are based on this manual split. In the second approach, we used the automatic splitting algorithm presented in Section 5.2.2. The resulting BDDs are identical to the manual split setting, and the subsequent runs of the EM algorithm converge to the same results. Hence when plotting against the iteration, the graphs are identical. The resulting ground theory is much larger and the initialization phase therefore takes 247 minutes. However, this is mainly due to the overhead for indexing, database access and garbage collection in the underlying Prolog[Santos Costa et al., 2011] system. Grounding and Clark's completion take only 6 seconds each, the term simplification step takes roughly 246 minutes, and the final splitting algorithm runs in 40 seconds. As we did not optimize the implementation of the term simplification, we see a big potential for improvement, for instance by tabling intermediate simplification steps. This affirmatively answers **Q3**.

5.3.2 Smokers

We set up an experiment on an instance of the *Smokers* dataset (cf. [Domingos and Lowd, 2009]) to answer the question

Q4 *Is LFI-Problog able to recover the parameters of the original model with a reasonable amount of data?*

Missing and incorrect values are two types of noise occurring in real-world data. While incorrect values can be compensated by additional data, missing values cause local maxima in the likelihood function. In turn, they cause the learning algorithm to yield parameters different from the ones used to generate the data. LFI-ProbLog attempts to compute the maximum likelihood parameters given some evidence. Hence the algorithm should be capable of recovering the parameters used to generate a set of interpretations. We analyze how the amount of required training data increases as the size of the model increases. Furthermore, we test for the influence of missing values on the results. We assess the quality of the learned model, that is, the difference to the original model parameters by computing the Kullback Leibler (KL) divergence a standard measure for differences of distribution [Duda et al., 2001] ProbLog allows for an efficient computation of this measure due to the independence of the probabilistic facts.

In this experiment, we use a variant of the "Smokers" model which can be represented in ProbLog as follows[6]:

[6]We used a tight version of this program as required by Clark's completion. We omit the details on that due to space reasons.

p_{si}::smokes_i(X,Y) % *person is influenced by one of their smoking friends*
p_{sp}::smokes_p(X) % *person starts smoking without external reason*
p_{cs} ::cancer_s(X). % *cancer is caused by smoking*
p_{cp} ::cancer(X). % *cancer without external reason*
smokes(X) :− friend(X,Y), smokes(Y), smokes_i(X,Y).
smokes(X) :− smokes_p(X).
cancer(X) :− smokes(X), cancer_s(X).
cancer(X) :− cancer_p(X).

We set the number of persons to 3, 4 and 5 respectively and sampled from the resulting models up to 200 interpretations each. From these datasets we derived new instances by randomly removing $10 - 50\%$ of the atoms. The size of an interpretation grows quadratically with the number of persons. The model, as described above, has an implicit parameter tying between ground instances of non-ground facts. Hence the number of model parameters does not change with the number of persons. To measure the influence of the model size, we therefore trained grounded versions of the model, where the grounding depends on the number of persons. For each dataset we ran LFI-ProbLog for 50 iterations of EM. Manual inspection showed that the probabilities stabilized after a few, typically 10, iterations. Figure 5.4 shows the KL divergence for 3, 4 and 5 persons respectively. The closer the KL divergence is to 0, the closer the learned model is to the original parameters. As the graphs show, the learned parameters approach the parameters of the original model as the number of training examples grows. Furthermore, the amount of missing values has little influence on the distance between the *true* and the learned parameters. Hence LFI-Problog is capable of recovering the original parameters with a reasonable amount of training data and it is robust against missing values. This affirmatively answers **Q4**.

5.4 Related Work

Existing parameter learning approaches for ProbLog [De Raedt et al., 2007], PRISM [Sato and Kameya, 2001], and SLPs [Muggleton, 1996] are mainly based on learning from entailment. For ProbLog, there exists a learning algorithm based on regression where each training example is a ground fact together with the target probability [Gutmann et al., 2008]. In contrast to LFI-ProbLog, this approach does *not* assume an underlying generative process; neither at the level of predicates nor at the level of interpretations. Sato and Kameya have contributed various interesting and advanced learning algorithms that have been incorporated in PRISM. However, all of them learn from entailment.

Ishihata et al. [2008] consider a parameter learning setting based on Binary Decision Diagrams (BDDs). In contrast to our work, they assume the BDDs to be given,

whereas LFI-ProbLog, constructs them in an intelligent way from evidence and a ProbLog theory. Ishihata *et al.* suggest that their approach can be used to perform learning from entailment for PRISM programs. This approach has been recently adopted for learning CP-Logic programs (cf. [Bellodi and Riguzzi, 2011]).

The BDDs constructed by LFI-ProbLog are a compact representation of all possible worlds that are consistent with the evidence. LFI-ProbLog estimates the marginals of the probabilistic facts in a dynamic programming manner on the BDDs. While this step is inspired by Ishihata et al. [2008], we tailored it towards the specifics of LFI-ProbLog, that is, we allow deterministic nodes to be present in the BDDs. This extension is crucial, as the removal of deterministic nodes can results in an exponential growth of the Boolean formulae underlying the BDD construction.

Riguzzi [2007] uses a transformation of ground ProbLog programs to Bayesian networks in order to learn ProbLog programs from interpretations. Such a transformation is also employed in the learning approaches for CP-logic [Vennekens et al., 2006]. The learning algorithm of CPT-L (Chapter 4) is closely related to LFI-ProbLog. However, CPT-L is targeted towards the sequential aspect of the theory, whereas we consider a more general settings with arbitrary theories. In Chapter 4, we assume full observability, which allows them to split the sequence into separate transitions. They build one BDD per transition, which is much easier to construct than one large BDD per sequence. Our splitting algorithm (cf. Sect 5.2.2) is capable of exploiting arbitrary independence.

Our approach can also be related to the work on knowledge-based model construction approaches in statistical relational learning such as BLPs [Kersting and De Raedt, 2007], PRMs [Getoor et al., 2001] and MLNs [Richardson and Domingos, 2006]. While the setting explored in this part is standard for the aforementioned formalisms, our approach has significant representational and algorithmic differences from the algorithms used in those formalisms. In BLPS, PRMs and CP-logic, each training example is typically used to construct a ground Bayesian network on which a standard learning algorithm is applied. Although the representation generated by Clark's completion is quite close to the representation of Markov Logic, there are subtle differences. While Markov Logic uses weights on clauses, we use probabilities attached to single facts. Our approach has a clear probabilistic semantics. The usage of BDDs to perform inference for MLNs is not completely apparent and seems to be an interesting research direction.

5.5 Conclusions

We have introduced a novel parameter learning algorithm from interpretations for the probabilistic logic programming language ProbLog. This has been motivated by the differences in the learning settings and applications of typical knowledge-based

model construction approaches and probabilistic logic programming approaches. The LFI-ProbLog algorithm tightly couples logical inference with a EM algorithm at the level of BDDs. We provide an empirical evaluation that demonstrates the applicability of the proposed algorithm to the types of problems commonly tackled by knowledge-based model construction approaches to statistical relational learning.

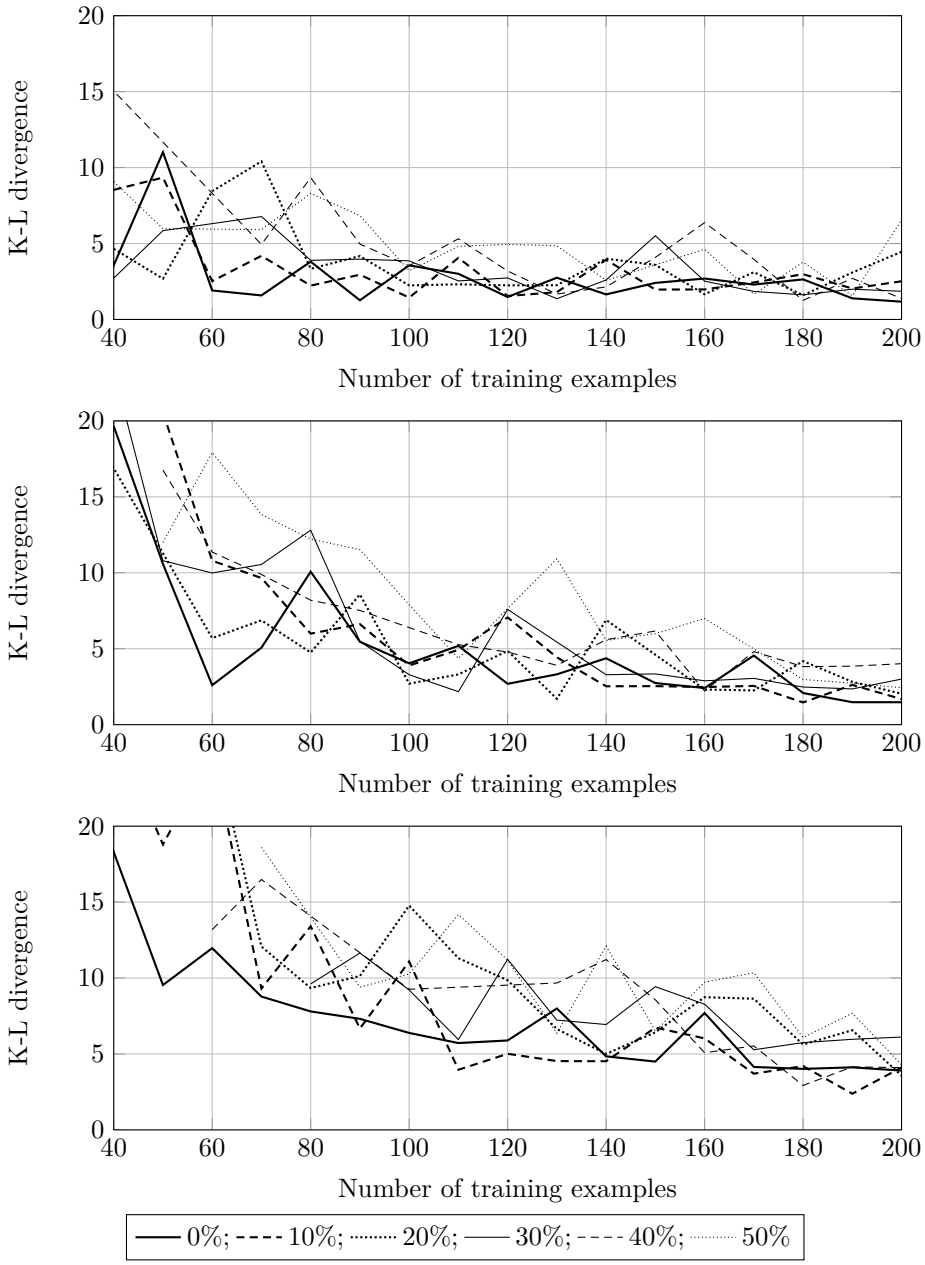


Figure 5.4: K-L divergence of the model trained using LFI-ProbLog on a Smokers data set with 3, 4 and 5 people (top, middle, bottom). The graphs represent different amounts of missing values in the training set (**Q4**).

Decision Making With ProbLog

6

This chapter shows how to make a rational decision by means of probabilistic programming. We introduce DTProbLog, which extends ProbLog with decisions and utilities. We study methods for exact and approximate inference.

Artificial intelligence is often viewed as the study of how to act rationally [Russell and Norvig, 2003]. The problem of acting rationally has been formalized within decision theory using the notion of a **decision problem**. In this type of problem, one has to choose actions from a set of alternatives, given a utility function. The goal is to select the strategy (set or sequence of actions) that maximizes the utility function. While the field of decision theory has devoted a lot of effort to deal with various forms of knowledge and uncertainty, there are so far only a few approaches that are able to cope with both uncertainty and rich logical or relational representations (see [Poole, 1997; Nath and Domingos, 2009; Chen and Muggleton, 2009]). This is surprising, given the popularity of such representations in the field of statistical relational learning [Getoor and Taskar, 2007; De Raedt et al., 2008].

To alleviate this situation, we introduce a framework combining ProbLog [Kimmig et al., 2008; De Raedt et al., 2007], with elements of decision theory. The resulting

This chapter builds on [Van den Broeck et al., 2010]

probabilistic programming language DTProblog (Decision-Theoretic ProbLog) is able to elegantly represent decision problems in complex relational and uncertain environments. A DTProbLog program consists like a ProbLog program of a set of definite clauses, and a set of probabilistic facts. In addition it consists of a set of decision facts, specifying which decisions are to be made, and a set of utility attributes, specifying the rewards that can be obtained. Further key contributions of this chapter include the introduction of an exact algorithm for computing the optimal strategy as well as a scalable approximation algorithm that can tackle large decision problems. These algorithms adapt the BDD based inference mechanism of ProbLog. While DTProbLog's representation and spirit are related to those of ICL [Poole, 1997], SLPs [Chen and Muggleton, 2009] , and MLNs [Nath and Domingos, 2009], its inference mechanism is distinct in that it employs state-of-the-art techniques using decision diagrams for computing the optimal strategy exactly; cf. the related work section ear or a more detailed comparison.

This chapter is organized as follows: in Section 6.1, we introduce DTProbLog and its semantics; Section 6.2 discusses inference and how to find the optimal strategy for a DTProbLog program; Section 6.3 evaluates the system experimentally and Section 6 describes related work; Finally, we conclude in Section 7.

6.1 Decision-Theoretic ProbLog

A ProbLog program consists of background knowledge $\mathcal{BK}$ and a set of probabilistic facts $\mathcal{F}_l$ labeled with their probabilities. To be able to encode decision problems we additionally need to be able to represent decisions and utilities. This extension is analogous to extending Bayesian networks to influence diagrams. Therefore DTProbLog program also contains a set of decision facts $\mathcal{D}$ and utility attributes $\mathcal{U}$, which we will define in turn.

6.1.1 Decisions and Strategies

Decision variables are represented by facts, and so, by analogy with the set of probabilistic facts $\mathcal{F}$, we introduce $\mathcal{D}$, the set of **decision facts**. We use the label ? as in $? :: d$ to denote that d is a decision fact. The difference between decisions and probabilistic facts, is that for the latter the environments samples the truth value, whereas this choice is done for the former by the user or agent.In [Van den Broeck et al., 2010] a strategy assigns a probability to decision facts. This means a strategy defines a distribution over ProbLog programs, whereas we take the view of two actors here.

This choice, called **strategy**, is a function $\sigma : \mathcal{D} \to \mathcal{P}(\mathcal{D})$, which maps a set of decision facts to a one of its subset, representing the true decisions. In analogy to

$\mathcal{F}_\omega$ we define $\mathcal{D}_\sigma = \sigma(\mathcal{D})$. Taking into account the decision facts $\mathcal{D}$ and the strategy σ, we can define the possible worlds, where a query succeeds (cf. Equation (3.1), page 35):

$$[\![q|\sigma]\!] := \{\mathcal{F}_\omega \subseteq \mathcal{F} \mid \mathcal{F}_\omega \cup \mathcal{BK} \cup \mathcal{D}_\sigma \models q\},$$

which allows us to define the success probability of query given a strategy σ as:

$$P([\![q|\sigma]\!]) := \sum_{\mathcal{F}_\omega \in [\![q|\sigma]\!]} P_{\mathcal{F}}(\mathcal{F}_\omega).$$

Abusing notation we will use $\sigma(\mathcal{DT})$ to denote ProbLog program $\mathcal{F}_l$ with the background knowledge $\mathcal{D}_\sigma \cup \mathcal{BK}$.

Example 6.1 *As a running example we will use the following problem of dressing for unpredictable weather:*

$$\mathcal{D} = \{\texttt{umbrella},\texttt{raincoat}\}$$

$$\mathcal{F}_1 = \{0.3 :: \texttt{rainy},\ 0.5 :: \texttt{windy}\}$$

$$\begin{aligned}\mathcal{BK} = \{\ & \texttt{broken\_umbrella} :- \texttt{umbrella}, \texttt{rainy}, \texttt{windy}.\\ & \texttt{dry} :- \texttt{not}(\texttt{rainy}).\\ & \texttt{dry} :- \texttt{umbrella}, \texttt{not}(\texttt{windy}).\ \%\ \textit{not}\ \texttt{windy}\ \textit{implies}\\ & \qquad\qquad\qquad\qquad\qquad\qquad \%\neg\texttt{broken\_umbrella}\\ & \texttt{dry} :- \texttt{raincoat}.\}\end{aligned}$$

There are two decisions to be made: whether to bring an `umbrella` *and whether to wear a* `raincoat`*. Furthermore,* `rainy` *and* `windy` *are probabilistic facts. The background knowledge describes when one gets wet and when one breaks the umbrella due to heavy wind. One can compute that the probability of* `dry` *is* 0.85*, when using the strategy* {`umbrella`}*. This is due to the explanations* {{`rainy`}, {}}*.*

6.1.2 Rewards and Expected Utility

The set $\mathcal{U}$ consists of **utility attributes** of the form $u_i \to r_i$, where u_i is a literal and r_i a reward for achieving u_i. The semantics is that whenever the query u_i succeeds, this yields a reward of r_i. Thus, utility attributes play a role analogous

to queries in ProbLog. The reward r_i is given only once, regardless for how many substitutions u_i succeeds. This is to keep consistency with the query probability.

Given a DTProbLog program $\mathcal{DT} = \mathcal{F} \cup \mathcal{BK} \cup \mathcal{U} \cup \mathcal{D}$, and a decision σ, the expected reward due to the attribute $a_i = (u_i \rightarrow r_i) \in \mathcal{U}$ is defined as

$$\mathrm{EU}_{\mathcal{DT}}(a_i|\sigma) = r_i \cdot P_{\mathcal{DT}}([\![u_i|\sigma]\!]). \tag{6.1}$$

Based on this definition we can now define the expected utility of a strategy, which corresponds to the cumulative rewards achieved. We assume that the rewards are additive, such that the expected utility can be defined as

$$\mathrm{EU}_{\mathcal{DT}}(\sigma) = \sum_{a_i \in \mathcal{U}} \mathrm{EU}_{\mathcal{DT}}(a_i|\sigma). \tag{6.2}$$

Whenever the program is clear from context, we will omit the index $\mathcal{DT}$ and write $\mathrm{EU}(\sigma)$.

Example 6.2 *We extend Example 6.1 with utilities:*

$$\mathcal{D} = \{\texttt{umbrella}, \texttt{raincoat}\}$$

$$\mathcal{F}_l = \{0.3 :: \texttt{rainy},\ 0.5 :: \texttt{windy}\}$$

$$\begin{aligned}\mathcal{BK} = \{&\texttt{broken\_umbrella} :\!- \texttt{umbrella}, \texttt{rainy}, \texttt{windy}.\\ &\texttt{dry} :\!- \texttt{not}(\texttt{rainy}).\\ &\texttt{dry} :\!- \texttt{umbrella}, \texttt{not}(\texttt{windy}).\\ &\texttt{dry} :\!- \texttt{raincoat}.\}\end{aligned}$$

$$\begin{aligned}\mathcal{U} = \{&\texttt{umbrella} \rightarrow -2, \qquad \texttt{raincoat} \rightarrow -20\\ &\texttt{broken\_umbrella} \rightarrow -40,\ \ \texttt{dry} \rightarrow 60\}\ ,\end{aligned}$$

Bringing an umbrella, breaking the umbrella or wearing a raincoat incurs a cost. Staying dry gives a reward. The expected utility of the strategy $\sigma = \{\texttt{umbrella}\}$ *is*

$$\begin{aligned}\mathrm{EU}(\sigma) = \mathrm{EU}(\{\texttt{umbrella}\}) &= -2 \cdot P([\![\texttt{umbrella}|\sigma]\!]) + 60 \cdot P([\![\texttt{dry}|\sigma]\!])\\ &\quad - 40 \cdot P([\![\texttt{broken\_umbrella}|\sigma]\!])\\ &= -2 \times 1 + 60 \times 0.85 - 40 \times 0.15 = 43\end{aligned}$$

Algorithm 5 Calculating the utility of a strategy

function UTILITY(strategy σ, program $\mathcal{DT} = \mathcal{F} \cup \mathcal{D} \cup \mathcal{BK} \cup \mathcal{U}$)
  $Utility := 0$
  $\mathcal{T}_\sigma := \sigma(\mathcal{DT})$ ▷ ProbLog program with probabilistic facts $\mathcal{F}$ and background knowledge $\mathcal{BK} \cup \sigma(\mathcal{D})$
  for all $(u_i \to r_i) \in \mathcal{U}$ **do**
    $E :=$ GENEXPLANATIONS$(u_i, \mathcal{T}_\sigma)$
    $p :=$ QUERYPROBABILITY(E)
    $Utility = Utility + p \cdot r_i$

6.2 Inference

A DTProbLog theory represents decision problems. The typical inference tasks for these kinds of problems are:

- **Expected utility** (Section 6.2.1): Given a DTProbLog theory $\mathcal{DT}$ and a strategy σ, calculate the expected utility $\mathrm{EU}_{\mathcal{DT}}(\sigma)$.
- **Optimal strategy** (Section 6.2.2): Given a DTProbLog theory, find the optimal strategy $\sigma^* = \mathrm{argmax}_\sigma(\mathrm{EU}_{\mathcal{DT}}(\sigma))$.

We will present algorithmic solutions to these tasks in turn.

6.2.1 Expected utility

The first problem that we will tackle is how to perform inference in DTProbLog, that is, how to compute the utility $\mathrm{EU}_{\mathcal{DT}}(\sigma)$ of a particular strategy σ in a DTProbLog program $\mathcal{DT}$. This is realized by first computing the success probability of all utility literals u_i occurring in $\mathcal{U}$ using the standard ProbLog inference mechanism. The overall utility $\mathrm{EU}(\sigma)$ can then be computed using Equation (6.2).

The calculation of the probability of a utility given a strategy σ is done by Algorithm 5. It is based on the algorithms in Section 3.2.1 and therefore generates the program $\sigma(\mathcal{DT})$ and then sums over all utilities.

Example 6.3 *Using Algorithm 2 (Section 3.2.1) and Figure 6.1 (left), it is easy to see that for the strategy of bringing an umbrella and not wearing a raincoat, the success probability of staying dry is* $\mathrm{P}([\![\texttt{dry}|\sigma]\!]) = 0.7 + 0.3 \cdot 0.5 = 0.85$ *and that* $\mathrm{EU}([\![\texttt{dry}|\sigma]\!]) = 60 \cdot 0.85 = 51$. *Using the BDD for* `broken_umbrella`*, we can calculate* $\mathrm{EU}(\sigma) = 51 + (0.15 \cdot (-40)) + (-2) = 43$.

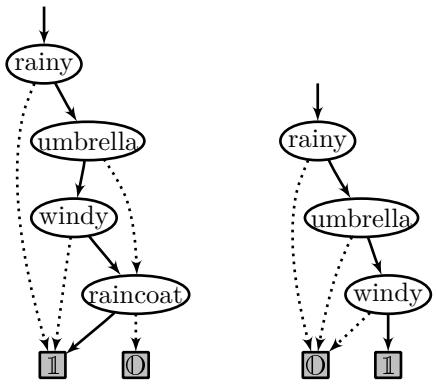


Figure 6.1: BDDs for `dry` (left) and `broken_umbrella` (right).

6.2.2 Solving Decision Problems

When faced with a decision problem, one is interested in computing the optimal strategy, that is, according to the maximum expected utility principle, finding σ^*:

$$\sigma^* = \text{argmax}_\sigma(\text{EU}(\sigma)).$$

This strategy is the **solution** to the decision problem. We will first introduce an exact algorithm for finding deterministic solutions and then outline two ways to approximate the optimal strategy.

Exact Algorithm

Our exact algorithm makes use of Algebraic Decision Diagrams (ADD) [Bahar et al., 1997] to efficiently represent the utility function $\text{EU}(\sigma)$.

The diagrams generated are:

1. $\text{BDD}_u(\mathcal{DT})$ representing $\mathcal{DT} \models u$ as a function of the probabilistic and decision facts in $\mathcal{DT}$. This procedure utilizes the standard ProbLog procedure to generate the set of all explanations. In the algorithm decision facts are treated like probabilistic facts,
2. $\text{ADD}_u(\sigma)$ representing $\text{P}(u|\sigma)$ as a function of σ,
3. $\text{ADD}_u^{\text{util}}(\sigma)$ representing $\text{EU}(u|\sigma)$ as a function of σ,
4. $\text{ADD}_{tot}^{\text{util}}(\sigma)$ representing $\text{EU}(\sigma)$ as a function of σ,

Algorithm 6 Finding the exact solution for $\mathcal{DT}$

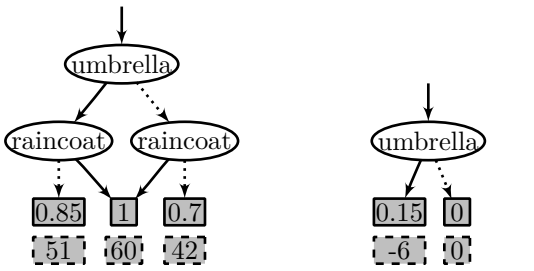
function ExactSolution((Theory $\mathcal{DT}$))
  $\text{ADD}^{\text{util}}_{tot}(\sigma) \leftarrow$ a 0-terminal
  for each $(u \to r) \in \mathcal{U}$ **do**
    $\text{BDD}_u(\mathcal{DT}) \leftarrow$ BinaryDD(u)
    $\text{ADD}_u(\sigma) \leftarrow$ ProbabilityDD($\text{BDD}_u(\mathcal{DT})$)
    $\text{ADD}^{\text{util}}_{u}(\sigma) \leftarrow r \cdot \text{ADD}_u(\sigma)$
    $\text{ADD}^{\text{util}}_{tot}(\sigma) \leftarrow \text{ADD}^{\text{util}}_{tot}(\sigma) \oplus \text{ADD}^{\text{util}}_{u}(\sigma)$
  let t_{max} be the terminal node of $\text{ADD}^{\text{u}}_{tot}(\sigma)$ with the highest utility
  let p be a path from t_{max} to the root of $\text{ADD}^{\text{u}}_{tot}(\sigma)$
  return the Boolean decisions made on p

function ProbabilityDD((BDD-node n))
  if n is the 1-terminal **then return** a 1-terminal
  if n is the 0-terminal **then return** a 0-terminal
  let h and l be the high and low children of n
  $\text{ADD}_h \leftarrow$ ProbabilityDD(h)
  $\text{ADD}_l \leftarrow$ ProbabilityDD(l)
  if n represents a decision d **then**
    return ite($d, \text{ADD}_h, \text{ADD}_l$)
  if n represents a fact with probability p **then**
    return $(p_n \cdot \text{ADD}_h) \oplus ((1 - p_n) \cdot \text{ADD}_l)$

These four diagrams map to the steps in the for-loop of Algorithm 6. The first step builds the BDD for the query u_i as described in Section 3. The difference is that the nodes representing decisions get marked as such, instead of getting a 0/1-probability assigned. In the second step, this BDD is transformed into an ADD using Algorithm 6, an adaptation of Algorithm 4. The resulting ADD contains as internal nodes only decision nodes and the probabilities are propagated into the leaves. The third step scales $\text{ADD}_u(\sigma)$ by the reward for u as in Equation (6.1).

Example 6.4 *Figure 6.2 shows the* $\text{ADD}_{dry}(\sigma)$ *and* $\text{ADD}_{broken\_umbrella}(\sigma)$*, constructed using Algorithm 6 from the BDDs in Figure 6.1. The former confirms that* $\text{P}([\![dry|\sigma]\!]) = 0.85$ *for the strategy given in Example 6.3. The transformation to* $\text{ADD}^{util}_{u}(\sigma)$ *is done by replacing the terminals by their dashed alternatives.*

Finally, in the fourth step, this ADD is added to the global sum $\text{ADD}^{\text{util}}_{tot}(\sigma)$ according to Equation (6.2), modeling the expected utility of the entire DTProbLog theory. From the final ADD, the globally optimal strategy σ^* is extracted by following a path from the leaf with the highest value to the root of the ADD. Because ADDs provide a compressed representation and efficient operations, the exact solution algorithm is able to solve more problems in an exact manner than could be done by naively enumerating all possible strategies.



(a) `dry` (b) `broken_umbrella`

Figure 6.2: $\mathrm{ADD}_{dry}(\sigma)$ and $\mathrm{ADD}_{broken\_umbrella}(\sigma)$. The alternative, dashed terminals belong to $\mathrm{ADD}_{u}^{\mathrm{util}}(\sigma)$.

Example 6.5 *Figure 6.3 shows* $\mathrm{ADD}_{tot}^{util}(\sigma)$*. It confirms that the expected utility of the strategy from Example 6.3 is 43. It turns out that this is the optimal strategy. For wearing a raincoat, the increased probability of staying dry does not outweigh the cost. For bringing an umbrella, it does.*

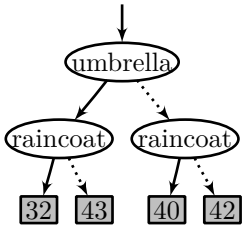


Figure 6.3: $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$ for $\mathrm{EU}(\sigma)$

Sound Pruning

Algorithm 6 can be further improved by avoiding unnecessary computations. The algorithm not only finds the best strategy, but also represents the utility of all possible strategies. Since we are not interested in those values, they can be removed from the ADDs when they become irrelevant for finding the optimal value. The idea is to keep track of the maximal utility that can be achieved by the utility attributes not yet added to the ADD. While adding further ADDs, all nodes of the intermediate ADD that can not yield a value higher than the current maximum can be pruned. For this, we define the maximal impact of a utility attribute to be

$$\mathrm{Im}(u_i) = \max(\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)) - \min(\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)),$$

where max and min are the maximal and minimal terminals. Before adding $\mathrm{ADD}_{u_i}^{\mathrm{util}}(\sigma)$ to the intermediate $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$, we merge all terminals from $\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)$ with a value below

$$\max(\mathrm{ADD}_{tot}^{\mathrm{util}}(\sigma)) - \sum_{j \geq i} \mathrm{Im}(u_j)$$

by setting their value to minus infinity. These values are so low that even in the best case, they will never yield the maximal value in the final ADD. By sorting the utility attributes by decreasing values of $\mathrm{Im}(u)$, even more nodes are removed from the ADD. This improvement still guarantees that an optimal solution is found. In the following subsections, we will show two improvements which will not have this guarantee, but are much faster. The two improvements can be used together or independently.

Local Search

Solving a DTProbLog program is essentially a function optimization problem for $\mathrm{EU}(\sigma)$ and can be formalized as a search problem in the strategy space. We apply a standard greedy hill climbing algorithm that searches for a locally optimal strategy. This way, we avoid the construction of the ADDs in steps 2-4 of Algorithm 6. The search starts with a random strategy and iterates repeatedly over the decisions. It tries to flip a decision, forming σ'. If $\mathrm{EU}(\sigma')$ improves on the previous utility, σ' is kept as the current best strategy. The utility value can be computed using Equation (6.2)) and Equation (6.1). To efficiently calculate $\mathrm{EU}(\sigma')$, we use the BDDs generated by the BinaryDD function of Algorithm 6. During the search, the BDDs can be kept fixed. Only the probability values for those BDDs that are effected by the changed decision have to be updated.
As this is essentially a standard greedy algorithm, the usual optimization techniques apply, such as random restarts, blocking of decisions where decisions are changed together, simulated annealing where randomly locally bad decisions are accepted, as well as other search algorithms such as genetic algorithms and random walks.

Approximative Utility Evaluation

The second optimization is concerned with the first step of Algorithm 6 that finds all proofs for the utility attributes. In large decision problems this quickly becomes intractable. A number of approximative inference methods exist for ProbLog [Kimmig et al., 2008], among which is the k-best approximation. The idea behind it is that, while there are many proofs, only a few contribute significantly to the total probability. It incorporates only those k proofs, with maximal probability, computing a lower bound on the success probability of the query. The required proofs are found using a *branch-and-bound* algorithm. Similarly, we use the k-best proofs for the utility attributes to build the BDDs and ADDs in the strategy solution algorithms. This reduces the run-time and the complexity of the diagrams. For sufficiently high values of k, the solution strategy found will be optimal.

6.3 Experimental Evaluation

The experiments were set up to answer the questions:

(Q1) Does the exact algorithm perform better than naively calculating the utility for all possible strategies?

(Q2) How does local search compare to the exact algorithm in terms of run-time and solution quality?

(Q3) What is the trade off between run-time and solution quality for different values of k, using the k-best proofs approximation?

(Q4) Do the algorithms scale to large, real-world problems?

To answer these questions we tested the algorithms on the viral marketing problem, a prime example of relational non-sequential decision making. The viral marketing problem was formulated by Domingos and Richardson [2001] and used in experiments with Markov Logic [Nath and Domingos, 2009]. Given a social network structure consisting of `trusts(a,b)` relations, the decisions are whether or not to market to individuals in the network. A reward is given for people buying the product and marketing to an individual has a cost. People that are marketed or that trust someone that bought the product may buy the product. In DTProbLog, this problem can be modeled as:

```
? :: market(P).

0.4 :: viral(P ,Q).

0.3 :: from_marketing(P).

buys(P) :- person(P),market(P), from_marketing(P).

buys(P) :- person(P),trusts(P,Q), buys(Q), viral(P,Q).

% the utilities for each person defined by means of quantification

∀P : person(P) [market(P) → −2.]

∀P : person(P) [buys(P) → 5]
```

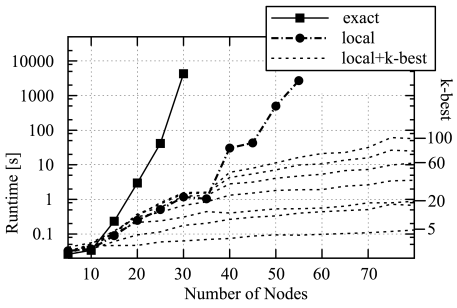


Figure 6.4: Run-Time of solving viral marketing in random graphs of increasing size. The values are averaged over three runs on four different graphs of the same size. Methods differ in the search algorithm and the number of proofs used.

The example shows the use of syntactic sugar in the form of templates for decisions and utility attributes. It is allowed to make decision facts and utility attributes conditional on a body of literals. For every substitution for which the body succeeds, a corresponding decision fact or utility attribute is constructed. We impose the restriction that these bodies can only depend on deterministic facts. For instance, in the example, there is one `market` decision for each person.

We implemented LFI-ProbLog in YAP Prolog [Santos Costa et al., 2011] and use CUDD [Somenzi, 2009] for the BDD operations and tested them on the viral marketing problem using a set of synthetic power law random graphs, known to resemble social networks [Barabasi and Bonabeau, 2003]. The average number of edges or trust relations per person was chosen to be 2. Figure 6.4 shows the run-time for different solution algorithms on graphs with increasing node count. For reproducibility, we start the local search algorithm from a zero-vector for σ and flip decisions in a fixed order.

This allows us to answer the first three questions: **(Q1)** While the exact algorithm is fast for small problems and guaranteed to find the optimal strategy, it becomes unfeasible on networks with more than 30 nodes. The 10-node problem is the final one solvable by the naive approach and takes over an hour to compute. Solving the 30-node graph in a naive manner would require over a billion inference steps, which is intractable. The exact algorithm clearly outperforms a naive approach. **(Q2)** Local search solves up to 55-node problems when it takes all proofs into account and was able to find the globally optimal solution for those problems where the exact algorithm found a solution. This is not necessarily the case for other decision problems with more deterministic dependencies. **(Q3)** After the 55-node point, the BDDs had to be approximated by the k-best proofs. For higher values of k, search becomes slower but is more likely to find a better strategy. For k larger than 20 the utility was within 2% of the best found policy for all problem sizes. To answer **(Q4)**, we experimented on a real world dataset of trust relations extracted

from the Epinions[1] social network website [Richardson and Domingos, 2002]. The network contains more than 75 thousand people that each trust 6 other people on average. Local search using the 17-best proofs finds a locally optimal strategy for this problem in 16 hours.

6.4 Related Work

Several AI-sub-fields are related to DTProbLog, either because they focus on the same problem setting or because they use compact data-structures for decision problems. Closely related is the *independent choice logic* (ICL) [Poole, 1997], which shares its *distribution semantics* [Sato, 1995] with ProbLog, and which can represent the same kind of decision problems as DTProbLog. Similar to DTProbLog being an extension of an existing language ProbLog, so have two related system been extended towards utilities recently. Nath and Domingos [2009] introduce *Markov logic decision networks* based on Markov logic networks and Chen and Muggleton [2009] extend *stochastic logic programs* (SLP) towards *decision-theoretic logic programs* (DTLP). The DTLP approach is close to the syntax and semantics of DTProbLog, although some restrictions are put on the use of decisions in probabilistic clauses. Chen and Muggleton also devise a parameter learning algorithm derived from SLPs. Nath and Domingos [2009] introduce *Markov logic decision networks* (MLDN) based on Markov logic networks. Many differences between MLNs and ProbLog exist and these are carried over to DTProbLog. Yet we are able to test on the same problems, as described earlier. Whereas DTProbLog's inference and search can be done both exact and approximative, MLDN's methods only compute approximate solutions. Some other formalisms too can model decision problems, e.g., IBAL [Pfeffer, 2001], and *relational decision networks* [Hsu and Joehanes, 2004]). Unlike DTProbLog, DTLPs, ICL and *relational decision networks* currently provide no implementation based on efficient data structures tailored towards decision problems and hence, no results are reported on a large problem such as the Epinions dataset. IBAL has difficulties representing situations in which properties of different objects are mutually dependent, like in the viral marketing example.

DTProbLog is also related to various works on Markov decision processes. In contrast to DTProbLog, these are concerned with sequential decision problems. Nevertheless, they are related in the kind of techniques they employ. For instance, for *factored Markov decision processes* [Boutilier et al., 2000], the SPUDD [Hoey et al., 1999] algorithm also uses ADDs to represent utility functions, though it cannot represent *relational* decision problems and is not based on probabilistic programming. On the other hand, there exist also *relational markov decision* processes [Boutilier et al., 2001] and *first-order Markov decision processes* [Boutilier

[1] `http://www.epinions.com/`

et al., 2001; Reiter, 2001]. FOMDPs model *sequential* decision problems, i.e. more general problems than we have described in this chapter. Several authors have developed different representation schemes and algorithms for solving relational by upgrading corresponding algorithms to the relational case [Kersting et al., 2004; Hölldobler et al., 2006]. Moreover, techniques including the development of compact *first-order decision diagrams* by Wang et al. [2008] and Sanner and Boutilier [2009]. While first-order decision diagrams are very attractive, they are not yet as well understood and well developed as their propositional counterparts.

6.5 Conclusions and Future work

A new decision-theoretic probabilistic logic programming language, called DTProbLog, has been introduced. It is a simple but elegant extension of the probabilistic Prolog ProbLog. Several algorithms for performing inference and computing the optimal strategy for a DTProbLog program have been introduced. This includes an exact algorithm to compute the optimal strategy using binary and algebraic decisions diagrams as well as two approximation algorithms. The resulting algorithms have been evaluated in experiments and shown to work on a real life application.

Perhaps the two most interesting questions for future work, relates this chapter to the previous chapters. The first question is concerned with decision making in sequential domains and the second with learning of DTProbLog programs.

The first of these questions is concerned with whether and how DTProbLog and its inference algorithms can be adapted for use in sequential decision problems and FOMDPs. While representing these is possible by either having time as extra argument, or along the lines of the CPT-L framework, there are two complications. First, care needs to be taken, when defining utilities. The semantics of DTProbLog is based on the distributions semantics, hence every query needs to fulfill the finite support condition. This implies that each utility is either achieved within a finite horizon, or not at all. Second, even if DTProbLog can represent such problems, the algorithmic solutions presented here, can not cope with such problems. For domains with finite horizon a investigation of the effectiveness of the algorithms still needs to be performed. This may motivate further modifications or extensions of DTProbLog's engine, such as tabling.

The second question is how to learn a DTProbLog program. LFI-ProbLog can already be used to learn the parameters in a restricted setting. If the strategy and the queries which generated the utilities are given LFI-ProbLog can directly be applied. A more general setting, where decisions are unobserved can be solved by assuming that the agent has been using the maximum expected utility policy, which can be computed by means of the algorithm presented. Incorporating the utility

value achieved as evidence – the current estimated values might be inconsistent with the observation – and learning the utilities is less obvious but provides an interesting research opportunity.

7 Summary and future work

This chapter concludes the thesis. It provides a summary of the contributions and points at some directions for future work.

7.1 Thesis Summary

The work presented in this thesis is situated in the field of statistical relational artificial intelligence. which aims at developing machines that are able to act in noisy, complex environments that consist of multiple objects and relations among them [Kersting et al., 2010]. Such domains are often characterized by the following four challenges

(chal1) to be able to deal with a large number of objects,

(chal2) to take into account how these objects relate to each other,

(chal3) to handle uncertainty in the model,

(chal4) and finally, to be able to efficiently deal with temporal sequences.

To act successful in such domains a machine needs to be able to make decisions and to learn.

Over the past years, researchers have developed a wide variety of formalisms to meet these challenges [Koller et al., 1997; Sato and Kameya, 1997; Poole, 1997; Jaeger, 1997; Friedman et al., 1999; Getoor et al., 2001; Pfeffer, 2001; Richardson and Domingos, 2006; Kersting and De Raedt, 2007; De Raedt et al., 2007; Poole, 2008].

While these formalisms can conceptually represent sequential problems, there are few empirical demonstrations that show that the available learning and inference algorithms are able to cope with dynamic domains (chal4). Nevertheless, results from propositional graphical models seem to suggest that specialized models or algorithms are required for sequential domains [Boyen and Koller, 1998]. One option is to use approximate methods [Kersting et al., 2009]. Another option is to use models specialized for representing sequences, for examples, LoHMM [Kersting et al., 2006] and TildeCRF [Gutmann and Kersting, 2006], however, these models are restricted to a single stochastic process. Therefore, they do not provide a solution to the second challenge (chal2) and makes, for example, modelling the dynamics of a social network impossible. The third option is to use models tailored towards a specific task, for example, sequence alignment [Karwath et al., 2008].

Another observation motivating our research is that the differences in representations as well as learning and inference algorithms among the different StaR-AI formalisms effects which domains and setting they are typically applied to. Some representations (e.g., BLPs, Markov logic, IBAL) are based on knowledge-based model construction, which compiles the query and/or evidence into a propositional graphical model, which allows to leverage existing propositional learning and inference algorithms. These approaches have been applied to a wide-variety of problems.

On the other hand, the representations that do not use knowledge based model construction, such as approaches based on logic programming, were typically concerned with calculating the probability of a single query. This focus on the so-called success probability limits their applicability. The learning methods, for example, are typically limited to the classical inductive logic programming tasks. On the positive side, probabilistic logic programming has the strong advantage in that its semantics is well understood from the viewpoint of statistics [Sato, 1995] and also from the knowledge representation point of view [Halpern, 1990; Vennekens, 2007].

Finally, decision making has been mainly studied from the perspective of relational Markov decision processes and relational reinforcement learning. This has resulted in restrictions similar to the ones for sequential probabilistic relational models. Namely, models limiting the dynamics of the domain to one single random process exist [Sanner, 2010], furthermore, they do not allow for factorizing decisions. The latter aspect has been studied before, but either without providing efficient algorithms [Poole, 1997; Pfeffer, 2001] or providing only approximate algorithms

based on greedy hill climbing search [Nath and Domingos, 2009].

This thesis introduced novel algorithms that are able to deal with the challenges outlined above (chal1-chal4) in an efficient manner. We have developed a formalism for representing complex relational processes that admits efficient learning and inference algorithms. Furthermore, our research has contributed algorithms which allows for learning from interpretations for probabilistic logic programming and decision making by means of probabilistic logic programming languages. These accomplishments increase the applicability of these formalisms. The following subsections summarize the contributions made in this dissertation.

7.1.1 Stochastic relational processes

The first contribution is concerned with representations, inference and learning for stochastic relational processes. We introduced CPT-L as a representation language. In contrast to other, more general SRL and PLL approaches, CPT-L emphasizes computational efficiency rather than expressivity. Most importantly, in CPT-L we assume the absence of hidden states. Efficient inference and parameter learning is possible in CPT-L by a partially-lifted inference algorithm. The algorithm aggregates all groundings of rules where the chosen head element is logically entailed into a joint factor during probabilistic inference, thereby significantly reducing the size of the resulting inference problem. Furthermore, we introduced an algorithm for the filtering task for cases where parts of a state are not observable. The algorithm efficiently estimates the hidden state in cases where the assumptions underlying CPT-L are not fully met. We systematically evaluated the proposed algorithms in several real-world domains.

7.1.2 Learning from interpretations

We introduced a novel algorithm for parameter learning from interpretations for the probabilistic logic programming language ProbLog. The LFI-ProbLog algorithm tightly couples logical inference with an EM algorithm at the level of BDDs. This learning algorithm is closely related to the one for CPT-L. First, it provides a motivation for the propositional logical formula generated by CPT-L, which can be seen as specialization of Clark's completion. Second, it generalizes EM by means of the BDD algorithm of CPT-L (but also the one by Ishihata et al. [2008]). The generalization allows for hidden and deterministic variables. Third, as in CPT-L, it splits sequences into transitions; LFI-ProbLog exploits certain independence to split training examples. In fact, LFI-ProbLog, when applied to translated CPT-L, would automatically rediscover its learning algorithm. Finally, similar to the partial lifted algorithm of CPT-L, LFI-Problog provides a probabilistic version of unit propagation. We provide an empirical evaluation which demonstrates the

applicability of the proposed algorithm to the types of problems commonly tackled by knowledge-based model construction approaches to statistical relational learning.

7.1.3 Decision making

We introduced the decision-theoretic probabilistic programming language DTProbLog in order to find the optimal solution to relational decision problems. It is a simple but elegant extension of the ProbLog formalism. We provided several algorithms for performing inference and computing the optimal strategy for a DTProbLog program. This included an exact algorithm to compute the optimal strategy using binary and algebraic decisions diagrams as well as two approximate algorithms. The resulting algorithms have been evaluated empirically and are shown to work in a real life application.

7.2 Summary

The contributions of this thesis are:

- We introduced CPT-L, which allows one to model stochastic relational processes. Efficient inference and parameter learning is made possible by deliberately restricting the expressivity of the model.
- We introduced LFI-ProbLog, a novel parameter learning approach for ProbLog programs.
- We introduced DTProbLog which allows to represent decision problems in a probabilistic programming language. We presented algorithms for inference and for finding the optimal hypothesis.

7.3 Future work

There are two major directions for future research. The first is to establish a closer connection between the different contributions. This is required to achieve a holistic StaR-AI system on the basis of probabilistic logic programs. Even though, it is in principle possible to combine the different contributions there are still open issues. The second direction is to extend the individual contributions to allow for wider applicability but also to integrate the techniques introduced herein with other formalisms.

7.3.1 Establishing closer connections...

...between CPT-L and LFI-ProbLog

One of the most interesting questions about the connection of CPT-L and LFI-Problog is concerned with hidden states. The learning algorithm of CPT-L does not allow for any hidden information except from the selections of the head elements for the applicable rules. However, the parameters of an HCPT-theory mapped into ProbLog can be learned using LFI-Problog. Still, the run-time of the algorithm, when used for such models, needs to be investigated.

Also in cases where the learning algorithm of CPT-L can be applied, the difference in run-time of the two algorithms needs further investigation. One of the most important difference between these two algorithms is the way they exploit independence. While CPT-L makes explicit assumptions about which independence hold, LFI-ProbLog is able to automatically detects which indepedencies hold in the data. Experiments (e.g., in the WebKB domain) have shown that discovering the independencies takes a significant amount of time.

The second, more subtle difference between those two algorithms is the difference between CPT-L's partially lifted inference and LFI-ProbLog's probabilistic unit propagation. While unit propagation captures more cases as it considers each grounding independently, the partial lifted inference is supposedly faster as it propagates the truth value of all groundings once time. Therefore, exploring how to use unit propagation within CPT-L deserves further investigation.

Third, the idea of the filtering algorithm of HCPT-L can be applied to Problog. Instead of building one BDD per transition, boundaries can be defined in terms of subsets of the Herbrand universe. These boundary sets can be used to block SLD-resolution. Instead of proving the current sub-goal it is added as node in the BDD and it also generates a BDD representing the proofs for this atom.

This allows to generate smaller BDDs, which can be used for optimal sampling in the sub-regions defined by the boundaries.

Finally, CPT-L encodes the transition model of a Markov model using CP-L clauses. Similarly, the transition model can be encoded, for example in ProbLog, where the equivalent restriction would be not observing any probabilistic facts, but also in any other SRL system. However, it is unclear whether there exist efficient inference algorithms for the ProbLog encoding.

...between CPT-L and DTProbLog

Also the connection between CPT-L and DTProbLog needs further exploration.

DTProbLog can modeling of sequential domains by adding an extra argument to each predicate to capture time.

The problem is that the finite support condition implies a "finite horizon condition". A solution, which is supported by the distribution semantics, is to define utilities using queries existentially quantify time. Still building a BDD for a query, which has an infinite proof is not possible. Therefore, alternative algorithms need to be explored.

But the cases, which can be handled exact or approximately, deserve investigation. The first case are domains featuring a finite horizon condition. The second case is when the probability of generating a reward is strictly monotonic decreasing. In this case the k-best algorithm can be used to impose an approximate version of the finite support condition.

...between LFI-ProbLog and DTProbLog

Finally the combination between LFI-Problog and DTProblog raises interesting research opportunities. LFI-ProbLog can be trivially applied in cases where the strategy and the queries which generated the reward are observed. This is due to the fact that a DTProbLog program conditioned on a strategy yields a ProbLog program.

Handling an unobserved strategy can be solved along the line of a structural EM algorithm. Assuming that the current DTProbLog programming is the correct one, the strategy a rational agent would use can be computed with the methods presented in this thesis. This suggest a three step procedure: (1) compute optimal strategy, (2) calculate expectations of the probabilistic facts in the Problog program conditioned on the strategy, and (3) maximize likelihood based on the expectation. However, it is unknown whether this procedure yields reasonable results and hence experimental evaluation is needed. Furthermore, it is unclear whether this procedure preserves the convergence guarantee of the underlying EM algorithm.

The case where the individual queries that generated the achieved total utility are unobserved is less trivial. The problem is that there may be no combination of rewards that generates the observed one. Adjusting the rewards (e.g., using regression) requires defining an error measure and it also may be necessary to regularize the rewards.

7.3.2 Extensions

The first extension lies at the heart of Star-AI, which is not only concerned with abstract representations, but also with reasoning on an abstract level. This kind of

reasoning or inference is known as lifted inference. While we presented first results for CPT-L by means of the partial lifted inference algorithm more elaborated methods exist [Poole, 2003; Milch et al., 2008; Kersting et al., 2009]. However, it is an open questions as to apply these methods to the models presented here. The inference algorithm for CPT-L but also LFI-ProbLog compiles the model and the evidence into a weighted CNF. This weighted CNF allows to use algorithms for lifted weighted model counting like, for example, counting belief propagation [Kersting et al., 2009] and first order knowledge compilation [Van den Broeck et al., 2011] instead of BDDs.

However, the ability to solve most the problems directly on the first order level comes at a price, namely much more complicated algorithms, which are in the general case not necessarily more efficient. We presented a partially lifted method for CPT-L, which lifts only simple cases though doing first order unit propagation. A similar idea that identifies the parts of the model where exact inference is possible and solves it on the first order level, which has been proposed for Markov logic networks [Jha et al., 2010].

A second extension is concerned with the combination of discrete distributions and continuous distributions. We recently achieved a first result [Gutmann et al., 2011b], which introduces the semantics of a language similar to CP-Logic, this language lacks efficient learning and inference algorithms. It is also worthwhile to investigate what constitutes a reasonable set of independence assumptions for the hybrid sequential setting.

As exact inference in hybrid models is prohibitively slow, a learning method using exact estimations of marginal distributions is infeasible. Therefore, approximate procedures need to be developed. Solving this problem probably requires developing a stochastic EM algorithm. It would also be interesting to apply such an algorithm to ProbLog and CPT-L. A faster learning algorithm will broaden the applicability of our algorithms to larger problems. Additional training data also might help compensate for the approximations made during learning and avoid overfitting. One algorithm which could serve as a basis for such studies has been proposed by Wingate et al. [2011]. This algorithm has been applied to two probabilistic programming languages: Church and probabilistic Matlab. However, there are many open details on how to apply it to probabilistic logic languages like ProbLog.

But not only the application of approximate methods to ProbLog is an open question but also how to use methods like LFI-ProbLog in other probabilistic programming languages such as Church. Current leaning in Church is based on Bayesian inference.

A first investigation of different methods for estimating the marginals of a ProbLog program has been developed [Fierens et al., 2011]. Additionally, this work demonstrated that it is possible to transform non-tight Prolog programs into tight programs. Another important contribution is that it proposes using a d-

DNNF representation instead of BDDs. BDDs are the main bottleneck of the parameter learning algorithms and using d-DNNF in their place has the potential to improve the scalability of our learning algorithms.

MCMC-based approaches are state-of-the-art methods for approximate inference in probabilistic programming. We are currently studying a method specifically for hybrid ProbLog programs. But this thesis seems to suggest an approach specifically for CP-Logic. The crucial component of the MCMC method is a Markov chain. The fourth chapter seems to suggest to use a CP-Logic program as CPT-L chain. This reduces a hard inference task into an easier task for fully observable CPT-L. Additionally the CPT-L framework could potentially be used in the context of relational reinforcement learning. In model based reinforcement learning, the agent constructs a representation of the dynamics of the environment [Kaelbling et al., 1996]. Previous works show how to apply probabilistic relational models in a relational reinforcement learning setting [Croonenborghs, 2009], but CPT-L provides a higher expressiveness than the models presented. It would thus be interesting to investigate whether RRL and CPT-L could be combined.

In this thesis text we only studied learning the parameters of probabilistic logic programs, a more general setting is structure learning. Here the goal is to a set of rules or clauses which accurately model the observed data. For ProbLog a first result was recently achieved [De Raedt and Thon, 2011]. But it tackled a limited setting that it learns how to model only a single target predicate. We are currently studying an extension which learns parameters and structure in parallel. The most interesting setting is learns complete programs, which includes the rules as well as the facts probabilities. But it is also the most complicated setting and needs further investigation.

A final question is how to adapt other SRL representations for coping with sequential domains. While results from propositional methods seem to indicate that exact algorithms will not scale very well, approximate methods could allow for efficient inference. It would be interesting to investigate whether techniques introduced for CPT-L and CP-logic could be adapted for use with other formalisms such as Markov Logic and BLPs, or programming languages such as Church and Blog. This may result in novel representations and possibilities for STAR-AI

Appendix 8

8.1 Translation of PPDDL theories

The **P**lanning **D**omain **D**efinition **L**anguage (PDDL) is widely-used language to specify planning problems [Younes and Littman, 2004] . It was developed for the International Planning Competition, and is based on the functional programming language LISP. It offers a well-standardized notation which allows an objective comparison of planning systems and reusing rules for different domains.

Probabilistic PDDL (PPDDL) is an extension which allows actions having probabilistic effects of the form (`probabilistic` p_1 e_1 $\ldots p_n$ e_n). As we are only interested in calculating probability and learn parameters, rewards do not get a special treatment.

Following [Rintanen, 2003] we define actions as follows: An **PPDDL action** $a = \langle \phi_a, e \rangle$ consists of a precondition ϕ_a, and an effect e. The action a is applicable in state s if $s \models \phi_a$. The effect e can be:

- $\top$ is the empty effect
- p and $\neg p$ are effects if p is a state variable, indicating that p is not true in the next state.

PPDDL description of the "bomb and the toilet" example

```
(define (domain bomb-and-toilet)
        (:requirements :conditional-effects :probabilistic-effects)
        (:predicates (bomb-in-package ?pkg) (toilet-clogged) (bomb-defused))
        (:action dunk-package :parameters (?pkg)
                 :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
                              (probabilistic 0.05 (toilet-clogged)))))
```

CPT-L description of the action

$$\texttt{rule1(Pkg)} : \texttt{bomb-defused} :: 1.0 \longleftarrow \texttt{action(dunk-package(Pkg),)}$$

$$\texttt{bomb-in-package(Pkg)}.$$

$$\texttt{rule2(Pkg)} : \texttt{toilet-clogged} :: 0.05 \vee \texttt{true} :: 0.95 \longleftarrow \texttt{action(dunk-package(Pkg))}.$$

Figure 8.1: The "Bomb and Toilet" example in PPDDL and the corresponding CPT-L description.

- $x \leftarrow f$ is an effect if x is a real valued state variable and f is a real valued function.
- $e_1 \wedge \ldots \wedge e_n$ are effects if the e_i's are effects, meant all effects take place
- $c \rhd e$ is a conditional effect if e is an effect and c is a formula, with the meaning that e takes place if c is true
- $p_1\ e_1 | \ldots | p_n\ e_n$ is an effect if e_i are effects and $p_i > 0$ and $\sum_i p_i = 1$. Where only one of the e_i take place with probability p_i.

PPDDL requires that the action model is consistent. A domain is called **consistent** if there are no two effects which can be applied at the same time and make a state variable true and false respectively. For example $b \wedge \neg b$ is inconsistent whereas $c \rhd b \wedge \neg c \rhd \neg b$ is consistent.

Furthermore we assume here that all actions are given in conditional normal form (CNF). That is, no conditional is nested into a probabilistic effect. For example the action

$$(0.1\ \top$$

$$\quad | \ 0.8\ (move(a,b) \wedge on(a,c)) \rhd (on(a,b) \wedge \neg on(a,c))$$

$$\quad | \ 0.1\ (move(a,b) \wedge on(a,c)) \rhd (on(a,Table) \wedge \neg on(a,c)))$$

is not in conditional normal form, whereas

$$(move(a,b) \wedge on(a,c)) \rhd (0.1\ \top$$

$$\mid 0.8\ (on(a,b) \wedge \neg on(a,c))$$

$$\mid 0.1\ ((on(a,Table) \wedge \neg on(a,c)))$$

is in conditional normal form.

From PPDDL to CPT-L

To use the algorithms from CPT-L for PPDDL, we first translate a set of PPDDL actions into a set of rules in CPT-L syntax. If there are no nested probabilistic effects this transformation corresponds to a one to one mapping of the probabilistic parameters. If this condition is violated the nested probabilistic effects have to be expanded. The transformation back can be achieved by solving the corresponding linear equation system. Two changes to the semantics of CPT-L are required for this transformation:

1. The implicit encoding of the *frame axiom*, this means atoms stay true in the next state, if no
2. *negative effects*, overrules the default frame axiom

We will show how this can be achieved after giving the transformation.

This allows us to transform all actions which have their effect in normal form $a = \langle \phi_a, (c_1 \rhd (h_{1,1} \wedge h_{1,k} \ldots \wedge c_i \rhd (h_{m,1} \wedge h_{m,l}))) \rangle$ into a corresponding set of rules as follows:

$$(h_{11} \wedge \ldots \wedge h_{1,k}) : p_1 \mid \ldots \mid (h_{m,l} \wedge \ldots \wedge h_{m,l}) : p_m \longleftarrow a, \phi_a, c_1$$

$$e_2 \longleftarrow a, \phi_a, c_2$$

$$\vdots$$

$$e_i \longleftarrow a, \phi_a, c_n$$

Note that multiple rules might originate from the same action.

Example 8.1 *The blocks world example compiles to*

$$(on(A,B) \wedge \neg on(A,C) : 0.8) \vee (\top : 0.1)$$

$$\vee\ (on(A,table) \wedge \neg on(A,C) : 0.1)$$

$$\longleftarrow \underbrace{move(A,B)}_{action}, \underbrace{free(A), free(B), on(A,C)}_{precondition}.$$

Handling numeric effects and axioms can easily be achieved using logic programming in the background knowledge. Given an action a, a domain model defines a distribution over possible successor states, $P(I_{t+1} \mid I_t, a)$, similar to CPT-L. The only difference is that the notation of applicability has to incorporate the action chosen, therefore $P(I_{t+1} \mid I_t, a) = P(I_{t+1} \mid I_t \cup \{a\})$

Due to the extensions to CPT-L the calculation of the successor probability needs to be slightly changed. Given an action a, a domain model defines a distribution over possible states, $P(I_{t+1}|I_t, a)$ in the following way. Let $\mathbf{R}_{t,a} = \{r_1, ..., r_k\}$ denote the set of all ground rules applicable in the current state $I_t \cup a$. For each ground rule applicable in I_t one head element will affect the transformation from I_t into I_{t+1}. More formally, a *selection* σ is a mapping from rules r_i to indices j_i denoting that head element $h_{ij_i} \in head(r_i)$ is selected. In the stochastic process to be defined, I_{t+1} is a possible successor for the state I_t if and only if there is a selection σ such that $I_{t+1} = (I_t \setminus \{h^-_{1\sigma(1)}, ..., h^-_{k\sigma(k)})\}) \cup \{h^+_{1\sigma(1)}, ..., h^+_{k\sigma(k)})\}.$, where the $h^+_{i\sigma(i)}$ corresponds to the positive and $h^-_{j\sigma(j)}$ to the negative effects of the corresponding head elements. We say that σ *yields* I_{t+1} from I_t, denoted $I_t \xrightarrow{\sigma} I_{t+1}$, and define

$$P(I_{t+1}|I_t, a) := \sum_{\sigma: I_t \xrightarrow{\sigma} I_{t+1}} P(\sigma) = \sum_{\sigma: I_t \xrightarrow{\sigma} I_{t+1}} \Big(\prod_{(r_i, j_i) \in \sigma} p_{j_i} \Big) \tag{8.1}$$

where p_{j_i} is the probability associated with head element h_{ij_i} in r_i. As for other Markov processes, we can define the probability of a sequence $I_1, ..., I_T$ given an initial state I_0 by

$$P(I_1, ..., I_T) := \prod_{t=0}^{T} P(I_{t+1} \mid I_t). \tag{8.2}$$

This defines a distribution over all possible sequences of interpretations of length T.

8.2 Proof of Theorem 4.2

Theorem 4.2 *Let $\mathcal{B}$ be a BDD resulting from the conversion of an inference problem $P(I_{t+1} \mid I_{[0,t]})$, annotated with upward and downward probabilities as defined above, and let*

$$\Gamma = \{\sigma \mid I_{[0,t]} \xrightarrow{\sigma} I_{t+1}\}$$

be the set of selections yielding I_{t+1}. Then

$$\alpha(root) \quad = \quad \sum_{\sigma \in \Gamma} P(\sigma)$$

$$= \quad P(I_{t+1} \mid I_{[0,t]}). \tag{4.7}$$

Proof Let $\mathcal{B}$ be a BDD graph structure resulting from an inference problem $p(I_{t+1} \mid I_{[0,t]})$, and let the nodes in $\mathcal{B}$ be annotated with upward and downward probabilities as outlined in Section 4.2.2. Let $\mathcal{N}$ and $\mathcal{E}$ denote the nodes and edges in $\mathcal{B}$. To every edge $E \in \mathcal{E}$ we associate a weight $P(E)$ with

$$P(E) = \begin{cases} P(c \mid r): & E \text{ corresponds to a positive branch} \\ 1: & E \text{ corresponds to a negative branch} \end{cases}$$

where $r.c$ is the Boolean variable associated with the node N from which E originates, and $P(c \mid r)$ the probability of choosing head element c in rule r. A (directed) path R in $\mathcal{B}$ is a sequence $N_1 E_1 \ldots N_k E_k N_{k+1}$ with $E_i \in \mathcal{E}$ and $N_i \in \mathcal{N}$, and we always go downward in the BDD. We define the weight of a path as

$$P(R) = \prod_{i=1}^{k} P(E_i), \tag{8.3}$$

and denote by $\mathcal{R}(N)$ the set of all paths from a node $N \in \mathcal{N}$ to the 1-terminal. We first show that

$$\forall N \in \mathcal{N} : \alpha(N) = \sum_{R \in \mathcal{R}(N)} P(R), \tag{8.4}$$

by induction over the level of a node in $\mathcal{B}$.

Base Case: We need to show Equation (8.4) for the terminal nodes. If N is the 1-terminal, the (trivial) path $R = N$ is the only element of $\mathcal{R}(N)$, with $P(R) = 1$ according to Equation (8.3). Thus, Equation (8.4) holds. If N is the 0-terminal node, then $\mathcal{R}(N) = \varnothing$, thus $\sum_{R \in \mathcal{R}(N)} P(R) = 0$, and Equation (8.4) holds as well.

Induction: Let $N \in \mathcal{N}$ denote a non-terminal node, and let $r.c$ denote its associated Boolean variable. Let E^+ and E^- denote the positive and negative branch originating from N, and N^+ and N^- the corresponding child nodes. A path $R \in \mathcal{R}(N)$ either runs through E^+ or E^-. In the first case, we have $R = NE^+R'$ with $R' \in \mathcal{R}(N^+)$, and $P(R) = P(c \mid h)P(R')$. In the second case, we have $R = NE^-R'$ with $R' \in \mathcal{R}(N^-)$, and $P(R) = P(R')$. Thus,

$$\sum_{R \in \mathcal{R}(N)} P(R) = P(c \mid r) \sum_{R \in \mathcal{R}(N^+)} P(R) + \sum_{R \in \mathcal{R}(N^-)} P(R).$$

From the inductive assumption it follows that

$$\sum_{R \in \mathcal{R}(N)} P(R) = P(c \mid r)\alpha(N^+) + \alpha(N^-)$$

$$= \alpha(N),$$

completing the proof of Equation (8.4). Recall that there is a one-to-one correspondence between a selection σ yielding I_{t+1} and a path R from the root to the 1-terminal. Considering Equation (4.1) and Equation (8.3), we also see that $P(\sigma) = P(R)$. Thus,

$$\begin{aligned}\alpha(root) &= \sum_{R \in \mathcal{R}(root)} P(R) \\ &= \sum_{\sigma \in \Gamma} P(\sigma) \\ &= P(I_{t+1} \mid I_{[0,t]}),\end{aligned}$$

completing the proof of Theorem 4.2. ∎

8.3 Proof of Theorem 4.3

Before proving Theorem 4.3 we will prove the following lemma:

Lemma 8.1 *Let $\mathcal{B}$ be a BDD graph structure resulting from an inference problem $p(I_{t+1} \mid I_{[0,t]})$, let the nodes in $\mathcal{B}$ be annotated with upward and downward probabilities as outlined in Section 4.2.2, and let $N_1, \ldots, N_k$ denote all nodes at a given level n in the BDD. Then it holds that*

$$P(I_{t+1} \mid I_{[0,t]}) = \sum_{l=1}^{k} \beta(N_l)\alpha(N_l). \tag{8.5}$$

Proof We prove Lemma 8.1 by induction over the BDD level n.
Base case: $n = 0$**.** At level zero of the BDD, there is only a single node, namely the root node. Equation (8.5) follows from Theorem 4.2 and $\beta(root) = 1$ Equation (4.5):

$$\alpha(root) = \alpha(root)\beta(root) = P(I_{t+1} \mid I_{[0,t]}).$$

Induction: Assume that Equation (8.5) holds for level n. Let $r_i\theta.c_{ij}\theta$ denote the Boolean variable associated with level n in the BDD, and let $p = P(r_i.c_{ij})$ be the corresponding probability. Let N_l with $l = 1, \ldots, k$ denote all nodes at level n, and let N'_l with $l = 1, \ldots, k'$ denote all nodes at level $n + 1$. Let furthermore N_l^+ (N_l^-) denote the positive (negative) child node of N_l for $l = 1, \ldots, k$. We will refer

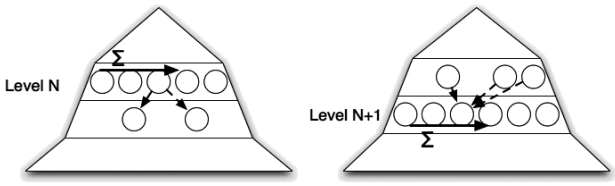


Figure 8.2: Inductive step in proof of $P(I_{t+1} \mid I_{[0,t]}) = \sum_{\sigma\in\Gamma} P(\sigma) = \sum_{l=1}^{k} \beta(N_l)\alpha(N_l)$.

by $pa^+(N'_l)$ ($pa^-(N'_l)$) to the subset of the nodes $N_1, \ldots, N_k$ which have N'_l as positive (negative) child node.

Starting from the inductive assumption, we now derive

$$P(I_{t+1} \mid I_{[0,t]}) = \sum_{l=1}^{k} \alpha(N_l)\beta(N_l)$$

$$= \sum_{l=1}^{k} \alpha(N_l^+)\beta(N_l)p + \sum_{l=1}^{k} \alpha(N_l^-)\beta(N_l) \tag{8.6}$$

$$= \sum_{l'=1}^{k'} \left[\sum_{N_l \in pa^+(N'_{l'})} \alpha(N'_{l'})\beta(N_l)p + \sum_{N_l \in pa^-(N'_{l'})} \alpha(N'_{l'})\beta(N_l) \right] \tag{8.7}$$

$$= \sum_{l'=1}^{k'} \alpha(N'_{l'})\beta(N'_{l'}). \tag{8.8}$$

Equation (8.6) follows from the definition of upward probabilities α. To derive Equation (8.7), we note that each edge from a node at level n either goes to a node at level $n+1$, or to the 0-terminal; because α(zero-terminal) $= 0$ the sums in Equation (8.6) and Equation (8.7) thus contain the same terms (see also Figure 8.2). Finally, Equation (8.8) follows from the definition of downward probabilities β. ∎

Theorem 4.3 *Let p_{ij} be the parameter associated with head element c_{ij} in rule r_i, let $\Delta = I_{[0,t]} \to I_{t+1}$ be a single transition, and let $r_i\theta \in \mathbf{R}_t$ denote a grounding of r_i applicable in $I_{[0,t]}$. Let $N_1, \ldots, N_k$ be all nodes in the BDD associated with the Boolean variable $r_i\theta.c_{ij}\theta$ resulting from the grounded rule $r_i\theta$, and let N_l^+ be the child on the positive branch of N_l. Then*

$$\mathbb{E}[\kappa_{ij}^{\theta} \mid \pi, \Delta] = \frac{1}{P(I_{t+1} \mid I_{[0,t]})} \sum_{l=1}^{k} \beta(N_l)p_{ij}\alpha(N_l^+). \tag{4.15}$$

Proof The nodes $N_1, \ldots, N_k$ associated with the variable $r_i\theta.c_{ij}\theta$ together form a level n of the BDD. As above let N_l^- denote the child on the negative branch of node N_l. Reconsidering Equation (8.6) in the proof of Lemma 8.1, we see that the probability of head element c_{ij} being selected is given by

$$P(\kappa_{ij}^\theta = 1 \mid \pi, \Delta) = \frac{\sum_{l=1}^k \alpha(N_l^+)\beta(N_l)p}{\sum_{l=1}^k \alpha(N_l^+)\beta(N_l)p + \sum_{l=1}^k \alpha(N_l^-)\beta(N_l)} \tag{8.9}$$

as the head element is chosen if and only if a node at level n is left through the positive branch. Because κ_{ij}^θ is a binary indicator,

$$\begin{aligned}
\mathbb{E}[\kappa_{ij}^\theta \mid \pi, \Delta] &= P(\kappa_{ij}^\theta = 1 \mid \pi, \Delta) \\
&= \frac{\sum_{l=1}^k \alpha(N_l^+)\beta(N_l)p}{\sum_{l=1}^k \alpha(N_l^+)\beta(N_l)p + \sum_{l=1}^k \alpha(N_l^-)\beta(N_l)} \\
&= \frac{1}{P(I_{t+1} \mid I_{[0,t]})} \sum_{l=1}^k \alpha(N_l^+)\beta(N_l)p \qquad (8.10)
\end{aligned}$$

where Equation (8.10) follows from the definition of downward probabilities β and Lemma 8.1. ∎

8.4 Proof of Theorem 5.1

Theorem 5.1 *For all ground probabilistic facts f_n and partial interpretations I_m*

$$\mathbb{E}_{\mathcal{T}}[\delta_{n,k}^m | I_m] = \begin{cases} \mathbb{E}_{\mathcal{T}^r(I_m)}[\delta_{n,k}^m | I_m] & \text{if } f_n \in \mathrm{dep}_{\mathcal{T}}(I_m) \\ p_n & \text{otherwise} \end{cases}$$

where $\mathcal{T}^r(I_m)$ is the interpretation-restricted ProbLog theory of $\mathcal{T}$ and p_n is the probability of the fact f_n. We assume that the naming of $\delta_{n,k}^m$ in $\mathbb{E}_{\mathcal{T}^r(I_m)}$ is such that the association of the facts in $\mathcal{F}_l$ and their groundings in $\mathcal{F}_l^r(I_m)$ is preserved.

Proof Due to the definition of $\mathrm{dep}_{\mathcal{T}}(I)$ all ground probabilistic facts $p_n :: f_n$ where $f_n \notin \mathrm{dep}_{\mathcal{T}}(I)$ are independent of I and we get $P_{\mathcal{T}}(f_n|I) = P_{\mathcal{T}}(f_n) = p_n$. In the case where $f_n \in \mathrm{dep}_{\mathcal{T}}(I)$ we have to show that:

$$P_{\mathcal{T}}(f_n|I) = \frac{P_{\mathcal{T}}(f_n, I)}{P_{\mathcal{T}}(I)} = \frac{P_{\mathcal{T}^r(I)}(f_n, I)}{P_{\mathcal{T}^r(I)}(I)} = P_{\mathcal{T}^r(I)}(f_n|I)$$

The following calculation is analog for numerator and denumerator:

$$P([\![\Phi(I)]\!]) = \sum_{\mathcal{F}_\omega \in [\![\Phi(I)]\!]} P_{\mathcal{F}}(\mathcal{F}_\omega)$$

$$= \sum_{\mathcal{F}_\omega \in [\![\Phi(I)]\!]} P_{\mathcal{F}}(\mathcal{F}_\omega \cap \mathrm{dep}_{\mathcal{T}}(I)) P_{\mathcal{F}}(\mathcal{F}_\omega \setminus \mathrm{dep}_{\mathcal{T}}(I))$$

as:

$$\begin{aligned}
[\![\Phi(I)]\!] &= \{\mathcal{F}_\omega \subseteq \mathcal{F} \mid \mathcal{F}_\omega \cap \mathcal{BK} \models \Phi(I)\} \\
&= \{\mathcal{F}_\omega \cap \mathcal{F}^r_\omega \mid (\mathcal{F}_\omega \subseteq \mathcal{F} \setminus \mathrm{dep}_{\mathcal{T}}(I)) \wedge \\
&\qquad\qquad (\mathcal{F}^r_\omega \subseteq \mathcal{F} \cap \mathrm{dep}_{\mathcal{T}}(I)) \wedge \mathcal{F}^r_\omega \cap \mathcal{BK}^r \models \Phi(I)\}
\end{aligned}$$

where the equivalence of $\mathcal{F}_\omega \cup \mathcal{BK} \models \Phi(I)$ and $\mathcal{F}^r_\omega \cup \mathcal{BK}^r \models \Phi(I)$ is due to the definition of $\mathrm{dep}_{\mathcal{T}}(I)$, hence we can rewrite

$$\begin{aligned}
P([\![\Phi(I)]\!]) &= \sum_{\mathcal{F}_\omega \in [\![\Phi(I)]\!]} P_{\mathcal{F}}(\mathcal{F}_\omega \cap \mathrm{dep}_{\mathcal{T}}(I)) P_{\mathcal{F}}(\mathcal{F}_\omega \setminus \mathrm{dep}_{\mathcal{T}}(I)) \\
&= \sum_{\mathcal{F}^r_\omega \in \{\mathcal{F}^r_\omega \in \mathcal{F}^r \mid \mathcal{F}^r_\omega \cap \mathcal{BK}^r \models \Phi(I)\}} P_{\mathcal{F}^r}(\mathcal{F}^r_\omega)
\end{aligned}$$

Bibliography

Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. Relational Markov models and their application to adaptive Web navigation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD02)*, KDD02, pages 143–152, 2002. ISBN 1-58113-567-X. DOI: 775047.775068. (p. 7)

Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors and effectors in the situation calculus. *Artif. Intell.*, 111(1-2): 171–208, 1999. (p. 3)

R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic Decision Diagrams and Their Applications. *Journal of Formal Methods in System Design*, 10:188–191, 1997. DOI: 10.1109/ICCAD.1993.580054. (p. 116)

Albert-Laszio Barabasi and Eric Bonabeau. Scale-free networks. *Scientific American*, 288(5):50–59, Mai 2003. DOI: 10.1038/scientificamerican0503-60. (p. 121)

Elena Bellodi and Fabrizio Riguzzi. EM over binary decision diagrams for probabilistic logic programs. Technical Report CS-2011-01, Dipartimento di Ingegneria, Università di Ferrara, Italy, 2011. (p. 107)

James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, New York, 1985. (p. 15)

Jeffrey P. Bezos. 2010 annual report, april 2010. URL http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9OTA4OTN8Q2hpbGRJRD0tMXxUeXBlPTM=&t=1. (p. 2)

Rahul Biswas, Sebastian Thrun, and Kikuo Fujimura. Recognizing activities with multiple cues. In *Human Motion - Understanding, Modeling, Capture and Animation, Second Workshop, Human Motion*, volume 4814 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2007. ISBN 978-3-540-75702-3. DOI: 10.1007/978-3-540-75703-0_18. (p. 72, 89)

Craig Boutilier and David Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the thirteenth national conference on Artificial intelligence (AAAI96)*, volume 2 of *AAAI'96*, pages 1168–1175. AAAI Press, 1996. ISBN 0-262-51091-X. (p. 23)

Craig Boutilier, Richard Dearden, and Moisacs Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49 – 107, 2000. ISSN 0004-3702. DOI: 10.1016/S0004-3702(00)00033-3. (p. 122)

Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic dynamic programming for first-order MDPs. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 690–700. Morgan Kaufmann, 2001. ISBN 1-55860-777-3. (p. 3, 8, 122)

Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI98)*, pages 33–42. Morgan Kaufmann, July 1998. (p. 7, 68, 126)

Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Pearson Education, Addison Wesley, 1990. 2nd Edition. (p. 25, 50)

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986. DOI: 10.1109/TC.1986.1676819. (p. 22, 24, 96)

Mark Chavira and Adnan Darwiche. Compiling bayesian networks using variable elimination. In Manuela M. Veloso, editor, *Proceedings of the 20th international joint conference on Artifical intelligence (IJCAI07)*, pages 2443–2449. Morgan Kaufmann Publishers Inc., January 2007. (p. 22)

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artifical Intelligence*, 172:772–799, April 2008. ISSN 0004-3702. DOI: 10.1016/j.artint.2007.11.002. (p. 18)

Jianzhong Chen and Stephan Muggleton. Decision-Theoretic Logic Programs. In *Lecture Notes in Computer Science, Inductive Logic Programming - 19th*

International Conference (ILP09), Lecture Notes in Computer Science, pages 226–233. Springer, September 2009. ISBN 978-3-642-13839-3. (p. 111, 112, 122)

E. Clarke, M. Fujita, P. Mcgeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. *Formal Methods Systems Design*, 10:149–169, April 1997. ISSN 0925-9856. DOI: 10.1023/A:1008647823331. (p. 24)

Mark Craven and Seán Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1):97–119, 2001. DOI: 10.1023/A:1007676901476. (p. 102)

Tom Croonenborghs. *Model-Assisted Approaches for Relational Reinforcement Learning*. PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering, September 2009. Bruynooghe, Maurice and Blockeel, Hendrik (supervisors). (p. 132)

James Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44:245–271, 2001. ISSN 0885-6125. DOI: 10.1023/A:1010924021315. (p. 8)

Adnan Darwiche. Bayesian networks. In Vladimir Lifschitz Frank van Harmelen and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 467–509. Elsevier, 2008. DOI: 10.1016/S1574-6526(07)03011-8. (p. 14)

Luc De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-20040-6. (p. 3, 91, 92)

Luc De Raedt and Kristian Kersting. Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):31–48, 2003. DOI: 10.1145/959242.959247. (p. 3)

Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In Shoham Ben-David, John Case, and Akira Maruoka, editors, *Algorithmic Learning Theory*, volume 3244 of *LNCS (Lecture Notes in Computer Science)*, pages 19–36. Springer Berlin / Heidelberg, 2004. DOI: 10.1007/978-3-540-30215-5_3. (p. 92)

Luc De Raedt and Ingo Thon. Probabilistic rule learning. In Paolo Frasconi and Francesca Alessandra Lisi, editors, *Proceedings of the 20th International Conference on Inductive Logic Programming (ILP10)*, volume 6489 of *LNCS (Lecture Notes in Computer Science)*, pages 47–58. Springer Berlin / Heidelberg, 2011. DOI: 10.1007/978-3-642-21295-6_9. (p. 11, 132)

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In Manuela M. Veloso, editor,

IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, pages 2462–2467, 2007. (p. 7, 32, 57, 91, 106, 111, 126)

Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors. *Probabilistic Inductive Logic Programming — Theory and Applications*, volume 4911 of *Lecture Notes in Artificial Intelligence*. Springer, 2008. (p. 46, 91, 111)

Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009. DOI: 10.2200/S00206ED1V01Y200907AIM007. (p. 32, 103, 104, 105)

Pedro Domingos and Matthew Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD01)*, pages 57–66. ACM, 2001. ISBN 1-58113-391-X. DOI: 10.1145/502512.502525. (p. 120)

Arnaud Doucet, Nando Defreitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice (Statistics for Engineering and Information Science)*. Springer, 1 edition, June 2001. ISBN 0-387-95146-6. DOI: 10.1198/tech.2003.s23. (p. 68, 69)

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2. edition, 2001. (p. 105)

Saso Dzeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52, 2001. (p. 3, 8)

Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory Practice Logic Programming*, 3:499–518, July 2003. ISSN 1471-0684. DOI: 10.1017/S1471068403001765. (p. 96)

François Fages. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science*, pages 51–60, 1994. (p. 96)

Jerome A. Feldman and Robert F. Sproull. Decision Theory and Artificial Intelligence II: The Hungry Monkey. *Cognitive Science: A Multidisciplinary Journal*, 1(2):158–192, 1977. DOI: 10.1207/s15516709cog0102_2. (p. 3)

Alan Fern. A Simple-Transition Model for Relational Sequences. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI05)*, pages 696–701. Morgan Kaufmann Publishers Inc., 2005. (p. 89)

Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. Inference in probabilistic logic programs using weighted CNF's. In *Proceedings of the Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 211–220, Corvallis, Oregon, 2011. AUAI Press. (p. 131)

Richard Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI71)*, pages 608–620, 1971. (p. 46)

Peter A. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994. (p. 3, 25)

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, pages 1300–1309. Morgan Kaufmann, 1999. ISBN 1-55860-613-0. (p. 6, 126)

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984. DOI: 10.1109/TPAMI.1984.4767596. (p. 18)

Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning*. The MIT Press, November 2007. (p. 3, 46, 91, 111)

Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In Sašo Džeroski and Nada Lavrač, editors, *Relational Data Mining*, pages 307–335. Springer Verlag, 2001. (p. 88, 91, 107, 126)

Zoubin Ghahramani. Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello"-Tutorial Lectures*, pages 168–197. Springer-Verlag, 1998. ISBN 3-540-64341-9. (p. 46, 54)

Zoubin Ghahramani and Michael I. Jordan. Factorial hidden markov models. *Machine Learning*, 29(2-3):245–273, 1997. DOI: 10.1023/A:1007425814087. (p. 54)

Noah Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: a language for generative models. In David A. McAllester and Petri Myllymäki, editors, *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI 2008)*, pages 220–229. AUAI Press, 2008. ISBN 0-9749039-4-9. (p. 7, 32)

Linda S. Gottfredson. Foreword to intelligence and social policy. *Intelligence*, 24 (1):1 – 12, 1997. ISSN 0160-2896. DOI: 10.1016/S0160-2896(97)90010-6. (p. 1)

Bernd Gutmann. *An inductive logic programming approach to statistical relational learning: Thesis*. PhD thesis, Department of Computer Science, K.U.Leuven, October 2011. (p. 32)

Bernd Gutmann and Kristian Kersting. TildeCRF: Conditional random fields for logical sequences. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML-2006)*, volume 4212 of *LNCS (Lecture Notes in Computer Science)*, pages 174–185. Springer, September 2006. DOI: 10.1007/11871842_20. (p. 7, 126)

Bernd Gutmann, Angelika Kimmig, Luc De Raedt, and Kristian Kersting. Parameter learning in probabilistic databases: A least squares approach. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *ECML 2008*, volume 5211 of *LNCS*, pages 473–488. Springer, 2008. (p. 8, 91, 104, 106)

Bernd Gutmann, Ingo Thon, and Luc De Raedt. Learning the parameters of probabilistic logic programs from interpretations. Technical Report CW 584, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, April 2010. (p. 91)

Bernd Gutmann, Ingo Thon, and Luc De Raedt. Learning the parameters of probabilistic logic programs from interpretations. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2011)*, volume 6911 of *LNCS (Lecture Notes in Computer Science)*, pages 581–596. Springer Berlin / Heidelberg, 2011a. DOI: 10.1007/978-3-642-23780-5_47. (p. 91)

Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11:663–680, 2011b. DOI: 10.1017/S1471068411000238. (p. 11, 32, 131)

Joseph Y. Halpern. An analysis of first-order logics of probability. *Journal of Artificial Intelligence Research*, 46(3):311–350, 1990. DOI: 10.1016/0004-3702(90)90019-V. (p. 126)

Eric A. Hansen and Zhengzhu Feng. Dynamic programming for POMDPs using a factored state representation. In *Proceedings of the 15th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, pages 130–139, April 2000. (p. 23)

Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI99)*, pages 279–288, August 1999. (p. 122)

Steffen Hölldobler, Eldar Karabaev, and Olga Skvortsova. Flucap: A heuristic search planner for first-order mdps. *Journal Artificial Intelligence Research*, 27: 419–439, 2006. DOI: 10.1613/jair.1965. (p. 123)

Ronald A. Howard and James E. Matheson. Influence diagrams. *Readings on the Princinples and Applications of Decision Analysis II (reprinted)*, 2(3):127–143, 1984/2005. DOI: 10.1287/deca.1050.0020. (p. 8, 21)

William Hsu and Roby Joehanes. Relational Decision Networks. In *Proceedings of the ICML Workshop on Statistical Relational Learning*, 2004. (p. 122)

Masakazu Ishihata, Yoshitaka Kameya, Taisuke Sato, and Shin-ichi Minato. Propositionalizing the EM algorithm by BDDs. In Filip Železný and Nada Lavrač, editors, *Proceedings of Inductive Logic Programming (ILP08), Late Breaking Papers*, pages 44–49, Prague, Czech Republic, September 2008. (p. 106, 107, 127)

Manfred Jaeger. Relational Bayesian Networks. In D. Geiger and P. P. Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI97)*, pages 266–273, Providence, Rhode Island, USA, 1997. Morgan Kaufmann. (p. 6, 61, 126)

F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001. (p. 14, 15, 21, 95)

Abhay Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted inference seen from the other side : The tractable features. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 973–981. 2010. (p. 131)

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996. (p. 132)

Andreas Karwath, Kristian Kersting, and Niels Landwehr. Boosting relational sequence alignment. In D. Gunopulos F. Giannotti, editor, *Proceedings of IEEE International Conference on Data Mining (ICDM08)*, pages 857–862, December 2008. ISBN 978-0-7695-3502-9. DOI: 10.1109/ICDM.2008.127. (p. 81, 82, 126)

Kristian Kersting. *An inductive logic programming approach to statistical relational learning: Thesis*. PhD thesis, Machine Learning Lab, CS Department, Albert-Ludwigs University, Freiburg, Germany, December 2006. (p. 3)

Kristian Kersting and Luc De Raedt. Bayesian Logic Programming: Theory and Tool. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*, pages 291–322. MIT Press, 2007. (p. 6, 46, 61, 88, 91, 107, 126)

Kristian Kersting and Luc De Raedt. Logical markov decision programs and the convergence of logical td(λ). In Rui Camacho, Ross D. King, and Ashwin Srinivasan, editors, *Inductive Logic Programming, 14th International Conference (ILP04)*, volume 3194 of *Lecture Notes in Computer Science*, pages 180–197.

Springer, 2004. ISBN 3-540-22941-8. DOI: 10.1007/978-3-540-30109-7_16. (p. 89)

Kristian Kersting, Martijn van Otterlo, and Luc De Raedt. Bellman goes relational. In Carla E. Brodley, editor, *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004)*, volume 69. ACM, 2004. DOI: 10.1145/1015330.1015401. (p. 123)

Kristian Kersting, Luc De Raedt, and Tapani Raiko. Logical hidden markov models. *J. Artif. Intell. Res. (JAIR)*, 25:425–456, 2006. DOI: 10.1613/jair.1675. (p. 7, 126)

Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In J. Bilmes A. Ng, editor, *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI09)*, June 2009. (p. 8, 126, 131)

Kristian Kersting, Stuart Russell, Leslie Kaelbling, Alon Halevy, Sriraam Natarajan, and Lilyana Mihalkova. Preface. In K. Kersting, S. Russell, L. Kaelbling, A. Halevy, S. Natarajan, and L. Mihalkova, editors, *Working Notes of the AAAI10 Workshop on Statistical Relational AI (StarAI)*. AAAI Press, 2010. (p. 2, 125)

Angelika Kimmig, Vítor Santos Costa, Ricardo Rocha, Bart Demoen, and Luc De Raedt. On the efficient execution of ProbLog programs. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 175–189. Springer Berlin / Heidelberg, 2008. DOI: 10.1007/978-3-540-89982-2_22. (p. 23, 111, 119)

Daphne Koller, David McAllester, and Avi Pfeffer. Effective bayesian inference for stochastic programs. In *Proceedings of the 14th national conference on artificial intelligence and 9th conference on Innovative applications of artificial intelligence (AAAI97/IAAI97)*, pages 740–747. AAAI Press, 1997. ISBN 0-262-51095-2. (p. 3, 126)

A. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Julius springer, Berlin, 1933. (p. 14)

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. (p. 7)

John E. Laird and Michael van Lent. Human-Level AI's Killer Application: Interactive Computer Games. In *Proceedings of the 7th National Conference on artificial intelligence and 12th conference on innovative applications of Artificial Intelligence*, pages 1171–1178. AAAI Press, 2000. ISBN 0-262-51112-6. (p. 45)

Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society Series B*, 50(2):157–224, 1988. DOI: 10.2307/2345762. (p. 18)

Andrey A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. 1907. (reprinted In Howard, E., editor, In Appendix A, *Dynamic Probabilistic Systems (Volume I: Markov Models)*, Appendix B, pages 552âĂŞ577. John Wiley & Sons, Inc., 1971.). (p. 7)

John McCarthy and Patrick J. Hayes. *Some philosophical problems from the standpoint of artificial intelligence*, pages 26–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-45-1. (p. 1)

Brian Milch, Bhaskara Marthi, Stuart J. Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 1352–1359. Professional Book Center, 2005. ISBN 0938075934. (p. 7)

Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulas. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1062–1068. AAAI Press, 2008. ISBN 978-1-57735-368-3. (p. 62, 64, 131)

Shin-ichi Minato, Ken Satoh, and Taisuke Sato. Compiling bayesian networks by symbolic probability calculation based on zero-suppressed bdds. In *Proceedings of the 20th international joint conference on Artifical intelligence (ICJAI07)*, pages 2550–2555. Morgan Kaufmann Publishers Inc., January 2007. (p. 22)

Tom M. Mitchell. *Machine Learnning*. McGraw-Hill, 1997. (p. 3)

Stephen Muggleton. Stochastic logic programs. In Luc De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 254–264. IOS Press, 1996. (p. 106)

Stephen Muggleton. Learning structure and parameters of stochastic logic programs. In Stan Matwin and Claude Sammut, editors, *Inductive Logic Programming*, volume 2583 of *Lecture Notes in Computer Science*, pages 198–206. Springer Berlin / Heidelberg, 2002. ISBN 3-540-00567-6. DOI: 10.1007/3-540-36468-4_13. (p. 8)

Paul Mutton. Inferring and visualizing social networks on internet relay chat. In *Proceedings of the Information Visualisation, Eighth International Conference*, pages 35–43. IEEE Computer Society, 2004. ISBN 0-7695-2177-0. DOI: 10.1109/IV.2004.75. (p. 72, 75)

Aniruddh Nath and Pedro Domingos. A Language for Relational Decision Theory. In Pedro Domingos and Kristian Kersting, editors, *International Workshop on Statistical Relational Learning (SRL-2009)*, Leuven, Belgium, 2-4 July 2009. (p. 8, 111, 112, 120, 122, 127)

E. Richard Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003. (p. 14)

Jennifer Neville and David Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–693, May 2007. (p. 6)

Ulf Nilsson and Jan Małuszyński. *Logic, programming, and Prolog*. Wiley New York, 1990. (p. 25, 28, 96)

Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. MIT Press, July 1994. ISBN 0-262-65040-1. (p. 15)

Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second National Conference on Artificial Intelligence (AAAI82)*, pages 133–âĂŞ136. AAAI Press., 1982. (p. 18)

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988. (p. 6, 16, 20)

Judea Pearl. An introduction to causal inference. Technical Report R-354, University of California, Los Angeles, Computer Science Department, 2009. (p. 17, 42)

Avi Pfeffer. IBAL: A probabilistic rational programming language. In Bernhard Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (ICJAI01)*, pages 733–740. Morgan Kaufmann Publishers, 2001. ISBN 1-55860-777-3. (p. 7, 8, 122, 126)

Martha E. Pollack. Intelligent technology for an aging population: The use of AI to assist elders with cognitive impairment. *AI Magazine*, 26(2):9–24, 2005. (p. 45)

David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, July 1997. DOI: 10.1016/S0004-3702(97)00027-1. (p. 3, 8, 32, 111, 112, 122, 126)

David Poole. First-order probabilistic inference. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI03)*, pages 985–991, 2003. (p. 62, 64, 131)

David Poole. The independent choice logic and beyond. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science (LNCS)*, pages 222–243. Springer, 2008. ISBN 978-3-540-78651-1. DOI: 10.1007/978-3-540-78652-8_8. (p. 7, 91, 126)

David Poole. Logic, probability and computation: Foundations and issues of statistical relational AI. In James P. Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference*, volume 6645 of *Lecture Notes in Computer Science*, pages 1–9. Springer, May 2011. ISBN 978-3-642-20894-2. DOI: 10.1007/978-3-642-20895-9_1. (p. 2)

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994. (p. 46)

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, February 1989. DOI: 10.1109/5.18626. (p. 7, 46, 54, 61)

Raghu Ramakrishnan and Johannes Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003. ISBN 978-0-07-115110-8. (p. 3)

Raymond Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001. ISBN 9780262182188. (p. 123)

Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD02)*, pages 61–70. ACM, 2002. ISBN 1-58113-567-X. DOI: 10.1145/775047.775057. (p. 122)

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006. DOI: 10.1007/s10994-006-5833-1. (p. 6, 46, 61, 86, 88, 91, 107, 126)

Fabrizio Riguzzi. Learning ground ProbLog programs from interpretations. In Donato Malerba, Annalisa Appice, and Michelangelo Ceci, editors, *Proceedings of the 6th International Workshop on Multi-relational Data Mining (MRDM07)*, pages 105–116, Warsaw, Poland, September 2007. (p. 107)

Jussi Rintanen. Expressive equivalence of formalisms for planning with sensing. In Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 185–194. AAAI, June 2003. ISBN 1-57735-187-8. (p. 133)

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003. (p. 1, 19, 50, 111)

Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf, 2010. (p. 8, 46, 126)

Scott Sanner and Craig Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, 2009. DOI: 10.1016/j.artint.2008.11.003. (p. 23, 123)

Vítor Santos Costa, David Page, and James Cussens. CLP($\mathcal{BN}$): Constraint logic programming for probabilistic knowledge. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*, pages 156–188. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-78651-1. DOI: 10.1007/978-3-540-78652-8. (p. 6)

Vítor Santos Costa, Ricardo Rocha, and Luís Damas. The YAP Prolog System. *Journal of Theory and Practice of Logic Programming*, 2011. To appear, arXiv:1102.3896v1 [cs.PL]. (p. 101, 105, 121)

T. Sato and Y. Kameya. PRISM: A language for symbolic-statistical modeling language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1330–1339. Morgan Kaufmann, 1997. (p. 3, 7, 32, 126)

Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In Leon Sterling, editor, *Proceedings of the Twelfth International Conference on Logic Programming (ICLP 1995)*, pages 715–729. MIT Press, 1995. ISBN 0-262-69177-9. (p. 7, 32, 34, 35, 92, 122, 126)

Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15: 391–454, 2001. DOI: 10.1613/jair.912. (p. 8, 91, 106)

Lawrence K. Saul and Michael I. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, 37:75–87, October 1999. ISSN 0885-6125. DOI: 10.1023/A:1007649326333. (p. 52)

Claude Elwood Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1948. (p. 23)

Fabio Somenzi. CUDD: CU Decision Diagram package release 2.4.2, 2009. URL http://vlsi.colorado.edu/~fabio/CUDD/. (p. 61, 101, 121)

Ingo Thon. Don't fear optimality: Sampling for probabilistic-logic sequence models. In *Lecture Notes in Computer Science, Inductive Logic Programming - 19th International Conference*, pages 226–233. Springer, September 2009. DOI: 10.1007/978-3-642-13840-9_{}22. (p. 45)

Ingo Thon, Niels Landwehr, and Luc De Raedt. A simple model for sequences of relational state descriptions. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD08)*, volume 5212 of *Lecture Notes in Computer Science (LNCS)*, pages 506–521. Springer Berlin / Heidelberg, 2008. DOI: 10.1007/978-3-540-87481-2_33. (p. 45, 72)

Ingo Thon, Bernd Gutmann, Martijn van Otterlo, Niels Landwehr, and Luc De Raedt. From non-deterministic to probabilistic planning with the help of statistical relational learning. In *Proceedings of the ICAPS Workshop on Planning and Learning,*, pages 23–30, August 2009. (p. 45)

Ingo Thon, Niels Landwehr, and Luc De Raedt. Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82:239–272, 2011. DOI: 10.1007/s10994-010-5213-8. (p. 45)

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. DOI: 10.1137/0208032. (p. 39)

Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, and Luc De Raedt. DTProbLog: A decision-theoretic probabilistic Prolog. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence,*, pages 1217–1222. AAAI Press, July 2010. (p. 111, 112)

Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In Toby Walsh, editor, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence,*, pages 2178–2185. AAAI Press/International Joint Conferences on Artificial Intelligence, 2011. (p. 131)

Joost Vennekens. *Algebraic and Logical Study of Constructive Processes in Knowledge Representation.* PhD thesis, Department of Computer Science, K.U.Leuven, 2007. (p. 126)

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. Representing causal information about a probabilistic process. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Logics in Artificial Intelligence, 10th European Conference, (JELIA 2006), Liverpool, UK*, volume 4160 of *Lecture Notes in Computer Science*, pages 452–464. Springer, 2006. ISBN 3-540-39625-X. DOI: 10.1007/11853886_37. (p. 7, 32, 46, 47, 61, 88, 107)

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009. DOI: 10.1017/S1471068409003767. (p. 40)

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. Embracing events in causal modelling: Interventions and counterfactuals in CP-logic. In Tomi Janhunen and Ilkka Niemela, editors, *Lecture Notes in Computer Science,*, pages 313–325. Springer, 2010. DOI: 0.1007/978-3-642-15675-5_{}27. (p. 42)

Chenggang Wang, Saket Joshi, and Roni Khardon. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31(1):431–472, 2008. DOI: 10.1613/jair.2489. (p. 123)

Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference (Springer Texts in Statistics)*. Springer, December 2003. ISBN 0387402721. (p. 3, 14)

Michael P. Wellman, John S. Breese, and Robert P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7:35–53, 1992. DOI: 10.1017/S0269888900006147. (p. 6)

David Wingate, Andreas Stuhlmueller, and Noah D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. *Proceedings of the 14th international conference on Artificial Intelligence and Statistics*, 2011. (p. 131)

Hakan L.S. Younes and Michael L. Littman. PPDDL1.0: The Language for the Probabilistic Part of IPC-4. In *Proceedings of the International Planning Competition*, 2004. (p. 46, 89, 133)

Luke S. Zettlemoyer, Hanna Pasula, and Leslie Pack Kaelbling. Learning Planning Rules in Noisy Stochastic Worlds. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI05)*, pages 911–918. AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X. (p. 46, 89)

Nevin Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994. (p. 18)

Publication List

Journal Articles

- Bernd Gutmann, Ingo Thon, Angelika Kimmig, Maurice Bruynooghe, and Luc De Raedt. *The magic of logical inference in probabilistic programming.* Theory and Practice of Logic Programming, 11:663–680, 2011. DOI: 10.1017/S1471068411000238
- Ingo Thon, Niels, Landwehr, Luc De Raedt *Stochastic relational processes: Efficient inference and applications.* Machine Learning, volume 82, issue 2, pages 239-272, 2011. DOI: 10.1007/s10994-010-5213-8
- Niels Landwehr, Bernd Gutmann, Ingo Thon, Luc De Raedt, and Matthai Philipose. *Relational transformation-based tagging for activity recognition.* Fundamenta Informaticae, 89(1):111–129, 2008.

Conference Papers

- Bernd Gutmann, Ingo Thon, and Luc De Raedt. *Learning the Parameters of Probabilistic Logic Programs from Interpretations.* In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, *European Conference on Machine Learning and Principles and Practices of Knowledge*

Discovery in Databases (ECML PKDD 2011), volume 6911 of LNCS (Lecture Notes in Computer Science), pages 581–596. Springer Berlin/Heidelberg, 2011. **Winner of the Best Paper Runner up Award in Machine Learning** (599 submissions). DOI: 10.1007/978-3-642-23780-5_47

- Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. *Inference in Probabilistic Logic Programs using Weighted CNF's*. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pages 211–220. AUAI Press, Corvallis, Oregon, 2011.

- De Raedt, Luc, Ingo Thon. *Probabilistic rule learning*, In Paolo Frasconi, Francesa Alessandra Lisi,*International Conference on Inductive Logic Programming (ILP)*, volume 6489 of LNCS, (Lecture Notes in Computer Science) pages 47-58, Springer Berlin/Heidelberg, 2011. DOI: 10.1007/s10994-010-5213-8

- Ingo Thon. *Don't fear optimality: Sampling for probabilistic-logic sequence models*. In Luc De Raedt, *Proceedings of the 19th International Conference on Inductive Logic Programming*, LNCS (Lecture Notes in Computer Science) volume 5989, pages 226-233, Springer, 2010 DOI: 10.1007/978-3-642-13840-9_22

- Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, Luc De Raedt. *DTProbLog: A decision-theoretic probabilistic Prolog*, In Maria Fox, David Poole, *Proceedings of the Twenty-fourth AAAI Conference on Artificial Intelligence* Atlanta, Georgia, USA, 11-15 July 2010, pages 1217-1222, AAAI Press

- Ingo Thon, Niels Landwehr, Luc De Raedt *A simple model for sequences of relational state descriptions*. In Walter Daelemans, Bart Goethals, and Katharina Morik, *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008)*, volume 5211 of LNCS (Lecture Notes In Computer Science), pages 473–488, September 2008. Springer Berlin/Heidelberg. DOI: `http://dx.doi.org/10.1007/978-3-540-87481-2_3310.1007/978-3-540-87481-2_33`

Workshop Papers

- Ingo Thon, Bernd Gutmann, Guy Van den Broeck. *Probabilistic programming for planning problems*. In Kristian Kersting, Stuart Russell, Leslie Pack Kaelbling, Alon Halevy, Sriraam Natarajan, Lilyana Mihalkova, *Statistical Relational AI workshop*, Atlanta, USA, 12 July 2010.

- Laura-Andreea Antanas, Bernd Gutmann, Ingo Thon, Kristian Kersting, Luc De Raedt. *Combining video and sequential statistical relational techniques to monitor card games.* In Christian Thurau, Kurt Diressens, and Olana Missura, *Proceedings of the ICML 2010 Workshop on Machine Learning and Games*, Haifa, Israel, 25 June 2010.

- Laura-Andreea Antanas, Bernd Gutmann, Ingo Thon, Kristian Kersting, and Luc De Raedt. *Combining video and sequential statistical relational techniques to monitor card games.* In Jan Ramon, Celine Vens, Kurt Driessens, Martijn Van Otterlo, and Joaquin Vanschoren, *The annual machine learning conference of Belgium and The Netherlands (BeneLearn 2010)*, Leuven, Belgium, 27-28 May 2010.

- Ingo Thon, Bernd Gutmann, Martijn van Otterlo, Niels Landwehr, Luc De Raedt. *From non-deterministic to probabilistic planning with the help of statistical relational learning*, *ICAPS 2009 - Workshop on Planning and Learning*, pages 23–30, Thessaloniki, 20 September 2009.

- Laura-Andreea Antanas, Martijn van Otterlo, Luc De Raedt, Ingo Thon. *Learning probabilistic relational models from sequential video data with applications in table-top and card games*, In Marieke van Erp, Herman Stehouwer Menno van Zaanen, *The annual machine learning conference of Belgium and The Netherlands (BeneLearn 2009)*, pages 105-106, Tilburg, 18-19 May 2009

- Maurice Bruynooghe, Broes De Cat, Jochen Drijkoningen, Daan Fierens, Jan Goos, Bernd Gutmann, Angelika Kimmig, Wouter Labeeuw, Steven Langenaken, Niels Landwehr, Wannes Meert, Ewoud Nuyts, Robin Pellegrims, Roel Rymenants, Stefan Segers, Ingo Thon, Jelle Van Eyck, Guy Van den Broeck, Tine Vangansewinkel, Lucie Van Hove, Joost Vennekens, Timmy Weytjens, and Luc De Raedt. *An exercise with statistical relational learning systems.* In Pedro Domingos and Kristian Kersting, *International Workshop on Statistical Relational Learning (SRL-2009)*, Leuven, Belgium, 2-4 July 2009.

- Ingo Thon. *Don't fear optimality: Sampling for probabilistic-logic sequence models (extended abstract).* In Luc De Raedt, Preliminary Proceedings of the 19th International Conference on Inductive Logic Programming (ILPâĂŹ09), Belgium, 1-4 July 2009,

- Luc De Raedt, Bart Demoen, Daan Fierens, Bernd Gutmann, Gerda Janssens, Angelika Kimmig, Niels Landwehr, Theofrastos Mantadelis, Wannes Meert, Ricardo Rocha, Vìtor Santos Costa, Ingo Thon, and Joost Vennekens. *Towards digesting the alphabet-soup of statistical relational learning.* In Daniel Roy, John Winn, David McAllester, Vikash Mansinghka, and Joshua Tenenbaum, editors, *Proceedings of the 1st Workshop on Probabilistic*

Programming: Universal Languages, Systems and Applications, Whistler, Canada, December 2008.

- Ingo Thon, Niels Landwehr, Luc De Raedt *A simple model for sequences of relational state descriptions.* In Filip Železný and Nada Lavrač, *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-2008), Late Breaking Papers*, pages 38–43, Prague, Czech Republic, September 2008.

- Ingo Thon, Niels Landwehr, Luc De Raedt. *CPT-L: An efficient model for relational stochastic processes.* In Samuel Kaski, S V N Vishwanathan, and Stefan Wrobel, editors, *Proceedings of the 6th International Workshop on Mining and Learning with Graphs (MLG 2008)*, Helsinki, Finland, 4-5 July 2008.

- Ingo Thon, Niels Landwehr, Luc De Raedt. *CPT-L: An efficient model for relational stochastic processes.* In Louis Wehenkel, Pierre Geurts, and Raphaël Marée, editors, *The annual machine learning conference of Belgium and The Netherlands (BeneLearn 2008)*, pages 27–28, Spa, Belgium, May 2008.

- Niels Landwehr, Bernd Gutmann, Ingo Thon, Matthai Philipose, and Luc De Raedt. *Relational transformation-based tagging for human activity recognition.* In João Gama, Mohamed Medhat Gaber, and Jesús Aguilar-Ruiz, *Proceedings of the International Workshop on Knowledge Discovery from Ubiquitous Data Streams (IWKDUS07)*, pages 83–94, Warsaw, Poland, September 2007.

- Niels Landwehr, Bernd Gutmann, Ingo Thon, Matthai Philipose, Luc De Raedt *Relational transformation-based tagging for human activity recognition.* In Donato Malerba, Annalisa Appice, and Michelangelo Ceci, *Proceedings of the 6th International Workshop on Multi-relational Data Mining (MRDM07)*, pages 81–92, Warsaw, Poland, September 2007.

- Ingo Thon, Kristian Kersting. Distributed relational state representations for complex stochastic processes, In Donato Malerba, Annalisa Appice, and Michelangelo Ceci, *Proceedings of the 6th International Workshop on Multi-relational Data Mining (MRDM07)*, pages 129-140

Technical Reports

- Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. *Inference in Probabilistic Logic Programs using Weighted CNF's.* Technical Report CW 607, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, June 2011.

- Luc De Raedt, Ingo Thon. *Probabilistic rule learning*, Technical Report CW580, Department of Computer Science, K.U.Leuven, Leuven, Belgium, April 2010
- Bernd Gutmann, Ingo Thon, and Luc De Raedt. *Learning the parameters of probabilistic logic programs from interpretations.* Technical Report CW 584, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, April 2010.

Curriculum vitae

Ingo Thon was born on December 8, 1978 in Freiburg, Germany. He went to school at the Technisches Gymnasium in Freiburg, from where he graduated (with "Allgemeiner Hochschulreife") in July 1999. After one year of compulsory civil service at the Institute of Medical Biometry and Medical Informatics at the University of Freiburg, he started studying computer science at the Albert-Ludwigs-University in Freiburg, Germany, in October 1999. In October 2003 he obtained a "Vordiplom", and in December 2006 a "Diplom" in computer science from this university.

In January 2007 he joined Luc De Raedt to the DTAI (Declarative Talen en Artificiële Intelligentie) group at the Katholieke Universiteit Leuven, Belgium and started work on a Ph.D. in the area of statistical relational learning, supervised by Luc De Raedt. In October 2011 he will defend his Ph.D. thesis on "Stochastic Relational Processes and Models, Learning and Reasoning" at the Katholieke Universiteit Leuven.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Declarative Languages and Artificial Intelligence
Celestijnenlaan 200A,
3001 Heverlee, Belgium

KATHOLIEKE UNIVERSITEIT LEUVEN

ASSOCIATIE K.U.LEUVEN