MORGAN & CLAYPOOL PUBLISHERS

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Luc De Raedt
Kristian Kersting
Sriraam Natarajan
David Poole

Synthesis Lectures on Artificial Intelligence and Machine Learning

Ronald J. Brachman and Peter Stone, *Series Editors*

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Synthesis Lectures on Artificial Intelligence and Machine Learning

Editors
Ronald J. Brachman, ***Yahoo! Labs***
William W. Cohen, ***Carnegie Mellon University***
Peter Stone, ***University of Texas at Austin***

Intelligent Autonomous Robotics: A Robot Soccer Case Study
Peter Stone
2007

Statistical Relational Artificial Intelligence: Logic, Probability, and Computation

Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole

www.morganclaypool.com

ISBN: 9781627058414 paperback
ISBN: 9781627058421 ebook

DOI 10.2200/S00692ED1V01Y201601AIM032

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Lecture #32
Series Editors: Ronald J. Brachman, *Yahoo! Labs*
William W. Cohen, *Carnegie Mellon University*
Peter Stone, *University of Texas at Austin*
Series ISSN
Print 1939-4608 Electronic 1939-4616

Statistical Relational Artificial Intelligence

Logic, Probability, and Computation

Luc De Raedt
KU Leuven, Belgium

Kristian Kersting
Technical University of Dortmund, Germany

Sriraam Natarajan
Indiana University

David Poole
University of British Columbia

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING #32

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

An intelligent agent interacting with the real world will encounter individual people, courses, test results, drugs prescriptions, chairs, boxes, etc., and needs to reason about properties of these individuals and relations among them as well as cope with uncertainty.

Uncertainty has been studied in probability theory and graphical models, and relations have been studied in logic, in particular in the predicate calculus and its extensions. This book examines the foundations of combining logic and probability into what are called relational probabilistic models. It introduces representations, inference, and learning techniques for probability, logic, and their combinations.

The book focuses on two representations in detail: Markov logic networks, a relational extension of undirected graphical models and weighted first-order predicate calculus formula, and Problog, a probabilistic extension of logic programs that can also be viewed as a Turing-complete relational extension of Bayesian networks.

KEYWORDS

probabilistic logic models, relational probabilistic models, lifted inference, statistical relational learning, probabilistic programming, inductive logic programming, logic programming, machine learning, Prolog, Problog, Markov logic networks

Contents

Preface

This book aims to provide an introduction that can help newcomers to the field to get started, to understand the state-of-the-art and the current challenges and be ready for future advances. It reviews the foundations of StarAI, motivates the issues, justifies some choices that have been made, and provides some open problems. Laying bare the foundations will hopefully inspire others to join us in exploring the frontiers and the yet unexplored areas.

The target audience for this book consists of advanced undergraduate and graduate students and also researchers and practitioners who want to get an overview of the basics and the state-of-the-art in StarAI. To this aim, Part I starts with providing the necessary background in probability and logic. We then discuss the representations of relational probability models and the underlying issues. Afterward, we focus first on inference, in Part II, and then on learning, in Part III. Finally, we touch upon relational tasks that go beyond the basic probabilistic inference and learning tasks as well as some open issues.

Researchers who are already working on StarAI—we apologize to anyone whose work we are accidentally not citing—may enjoy reading about parts of StarAI they are less familiar with.

We are grateful to all the people who contributed to the development of statistical relational learning and statistical relational AI. This book is made possible by you.

We also thank the reviewers for their valuable feedback and our co-authors, who accompanied us on our StarAI adventures, such as Laura Antanas, Udi Apsel, Babak Ahmadi, Hendrik Blockeel, Wolfram Burgard, Maurice Bruynooghe, David Buchman, Hung H. Bui, Peter Carbonetto, Alexandru Cocora, Fabrizio Costa, Michael Chiang, Walter Daelemans, Jesse Davis, Nando de Freitas, Kurt De Grave, Tinne De Laet, Bart Demoen, Kurt Driessens, Saso Dzeroski, Thomas G. Dietterich, Adam Edwards, Alan Fern, Daan Fierens, Paolo Frasconi, Roman Garnett, Amir Globerson, Bernd Gutmann, Martin Grohe, Fabian Hadiji, McElory Hoffmann, Manfred Jaeger, Gerda Janssens, Thorsten Joachims, Saket Joshi, Leslie Kaelbling, Andreas Karwath, Arzoo Katiyar, Seyed M. Kazemi, Angelika Kimmig, Jacek Kisynski, Tushar Khot, Stefan Kramer, Gautam Kunapuli, Chia-Li Kuo, Tobias Lang, Niels Landwehr, Daniel Lowd, Catherine A. McCarty, Theofrastos Mantadelis, Wannes Meert, Brian Milch, Martin Mladenov, Bogdan Moldovan, Roser Morante, Plinio Moreno, Marion Neumann, Davide Nitti, Phillip Odom, Jose Oramas, David Page, Andrea Passerini, Rui Pimentel de Figueiredo, Christian Plagemann, Tapani Raiko, Christopher Re, Kate Revoredo, Achim Rettinger, Ricardo Rocha, Scott Sanner, Vitor Santos Costa, Jose Santos-Victor, Erkal Selman, Rita Sharma, Jude W. Shavlik, Prasad Tadepalli, Nima Taghipour, Ingo Thon, Hannu Toivonen, Pavel Tokmakov, Sunna Torge, Marc Toussaint, Volker Tresp, Tinne Tuytelaars, Vincent Van Asch, Guy Van den Broeck, Martijn van Otterlo, Joost Vennekens, Jonas Vlasselaer, Zhao Xu, Shuo Yang, and Luke Zettlemoyer.

Thanks for all the encouragement and fun! Thanks to the StaRAI lab at Indiana for proofreading the book.

Last but not least, we also thank our families and friends for their patience and support. Thanks!

LDR and KK thank the European Commission for support of the project FP7-248258-First-MM. KK further thanks Fraunhofer Society, ATTRACT Fellowship "STREAM", the German Science Foundation, DFG KE 1686/2-1, as part of the DFG Priority Programme 1527, and the German-Israeli Foundation for Scientific Research and Development, GIF 1180/2011. SN thanks Army Research Office (ARO) grant number W911NF-13-1-0432 under the Young Investigator Program and the National Science Foundation grant no. IIS-1343940. LDR thanks the Research Foundation Flanders, and the KULeuven BOF fund for their support. DP thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) for ongoing support.

Luc De Raedt, Leuven, Belgium
Kristian Kersting, Dortmund, Germany
Sriraam Natarajan, Bloomington, USA
David Poole, Vancouver, Canada
February 2016

CHAPTER 1
Motivation

There are good arguments that an intelligent agent that makes decisions about how to act in a complex world needs to model its uncertainty; it cannot just act pretending that it knows what is true. An agent also needs to reason about individuals (objects, entities, things) and about relations among the individuals.

These aspects have often been studied separately, with models for uncertainty often defined in terms of features and random variables, ignoring relational structure, and with rich (logical) languages for reasoning about relations that ignore uncertainty. This book studies the integration of the approaches to reasoning about uncertainty and reasoning about individuals and relations.

1.1 UNCERTAINTY IN COMPLEX WORLDS

Over the last 30 years, **Artificial Intelligence** (AI) has evolved from being skeptical, even hostile, to the use of probability to embracing probability. Initially, many researchers were skeptical about **statistical AI** because probability seemed to rely on too many numbers and did not deal with the complexities of a world of individuals and things. But the use of probabilistic graphical models, exploiting probabilistic independencies, has revolutionized AI. The independencies specified in such models are natural, provide structure that enables efficient reasoning and learning, and allow one to model complex domains. Many AI problems arising in a wide variety of fields such as machine learning, diagnosis, network communication, computer vision, and robotics have been elegantly encoded and solved using probabilistic graphical models.

Meanwhile, there have also been considerable advances in **logical AI**, where agents reason about the structure of complex worlds. One aspect of this is in the semantic web and the use of ontologies to represent meaning in diverse fields from medicine to geology to the products in a catalogue. Generally, there is an explosive growth in the amount of heterogeneous data that is being collected in the business and scientific world. Example domains include biology and chemistry, transportation systems, communication networks, social networks, and robotics. Like people, intelligent agents should be able to deal with many different types of knowledge, requiring structured representations that give a more informative view of the AI task at hand.

Moreover, reasoning about individuals and relations is all about reasoning with regularities and symmetries. We lump individuals into categories or classes (such as "person" or "course") because the individuals in a category share common properties—e.g., there are statements that are true about all living people such as they breath, they have skin and two biological parents. Similarly for relations, there is something in common between Sam being advised by Professor

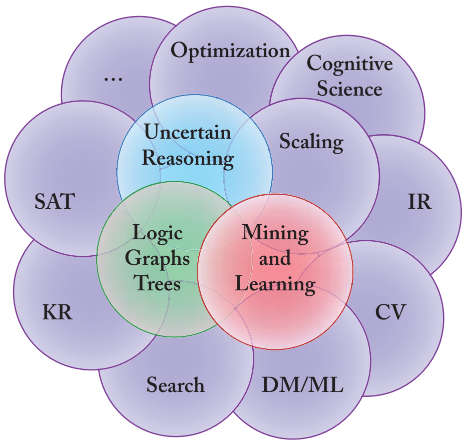


Figure 1.1: Statistical Relational Artificial Intelligence (StarAI) combines probability, logic, and learning and covers major parts of the AI spectrum.

Smith and Chris being advised by Professor Johnson; there are statements about publishing papers, working on a thesis and projects that are common among the "advised by" relationships. We would like to make predictions about two people about whom all we know may be only their advisory relationships. It is these commonalities and regularities that enable language to describe the world. Reasoning about regularities and symmetries is the foundation of logics built on the predicate calculus, which allows statements about all individuals.

Thus, to deal with the real world we actually need to exploit uncertainty, independencies, and symmetries and tackle a long standing goal of AI, namely unifying first-order logic—capturing regularities and symmetries—and probability—capturing uncertainty and independence. Predicate logic and probability theory are not in conflict with each other, they are synergistic. Both extend propositional logic, one by adding relations, individuals, and quantified variables, the other by allowing for measures over possible worlds and conditional queries. This may explain why there has been a considerable body of research in combining both of them over the last 25 years, evolving into what has come to be called **Statistical Relational Artificial Intelligence** (**StarAI**); see also Fig. 1.1:

> *the study and design of intelligent agents that act in worlds composed of individuals (objects, things), where there can be complex relations among the individuals, where the agents can be uncertain about what properties individuals have, what relations are true, what individuals exist, whether different terms denote the same individual, and the dynamics of the world.*

The basic building block of StarAI are relational probabilistic models—we use this term in the broad sense, meaning any models that combine relations and probabilities. They can be seen as combinations of probability and predicate calculus that allow for individuals and relations as well as probabilities. In building on top of relational models, StarAI goes far beyond reasoning, optimization, learning, and acting optimally in terms of a fixed number of features or variables, as it is typically studied in machine learning, constraint satisfaction, probabilistic reasoning, and other areas of AI. In doing so, StarAI has the potential to make AI more robust and efficient.

This book aims to provide an introduction that can help newcomers to the field get started, understand the state-of-the-art and the current challenges, and be ready for future advances. It reviews the foundations of StarAI, motivates the issues, justifies some choices that have been made, and provides some open problems. Laying bare the foundations will hopefully inspire others to join us in exploring the frontiers and the yet unexplored areas.

The target audience for this book consists of advanced undergraduate and graduate student as well as researchers and practitioners who want to get an overview of the basics and the state-of-the-art in StarAI. To this aim, Part I starts with providing the necessary background in probability and logic. We then discuss the representations of relational probability models and the underlying issues. Afterward, we focus first on inference, in Part II, and then on learning, in Part III. Finally, we touch upon relational tasks that go beyond the basic probabilistic inference and learning tasks as well as some open issues.

Researchers who are already working on StarAI—we apologize to anyone whose work we are accidentally not citing—may enjoy reading about parts of StarAI with which they are less familiar.

1.2 CHALLENGES OF UNDERSTANDING STARAI

Since StarAI draws upon ideas developed within many different fields, it can be quite challenging for newcomers to get started.

One of the challenges of building on top of multiple traditions is that they often use the same vocabulary to mean different things. Common terms such as "variable," "domain," "object," "relation," and "parameter" have come to have accepted meanings in mathematics, computing, statistics, logic, and probability, but their meanings in each of these areas is different enough to cause confusion. We will be clear about the meaning of these when using them. For instance, we follow the logic tradition and use the term "individuals" for things. They are also called "objects," but that terminology is often confusing to people who have been brought up with object-oriented programming, where objects are data structures and associated methods. For example, a person individual is a real person and not a data structure that encapsulates information about a person. A computer is not uncertain about its own data structures, but can be uncertain about what exists and what is true in the world.

Another confusion stems from the term "relational." Most existing datasets are, or can be, stored in relational databases. Existing machine learning techniques typically learn from datasets stored in relational datasets where the values are Boolean, discrete, ordinal, or continuous. However, in many datasets the values are the names of individuals, as in the following example.

student	*course*	*grade*
s_1	c_1	a
s_2	c_1	c
s_1	c_2	b
s_2	c_3	b
s_3	c_2	b
s_4	c_3	b
s_3	c_4	?
s_4	c_4	?

Figure 1.2: An example dataset that is not amenable to traditional classification.

Example 1.1 Consider learning from the dataset in Fig. 1.2. The values of the Student and the Course attributes are the names of the students (s_1, s_2, s_3 and s_4) and the courses (c_1, c_2, c_3 and c_4). The value of the grade here is an ordinal (a is better than b which is better than c). Assume that the task is to predict the grade of students on courses, for example predicting the grade of students s_3 and s_4 on course c_4. There is no information about course c_4, and students s_3 and s_4 have the same average (they both have one "b"); however, it is still possible to predict that one will do better than the other in course c_4. This can be done by learning how difficult each course is, and how smart each student is, given the data about students, the courses they take, and the grades they obtain. For example, we may learn that s_1 is intelligent, s_2 is not as intelligent, course c_2 is difficult and course c_3 is not difficult, etc. This model then allows for the prediction that s_3 will do better than s_4 in course c_4.

Standard textbook supervised learning algorithms that learn, e.g., a decision tree, a neural network, or a support vector machine (SVM) to predict *grade* are not appropriate; they can handle ordinals, but cannot handle the names of students and courses. It is the relationship among the individuals that provides the generalizations from which to learn. Traditional classifiers are unable to take into account such relations. This also holds for learning standard graphical models, such as Bayesian networks. These approaches make what can be seen as a single-table single-row assumption, which requires that each instance is described in a single row by a fixed set of features and all instances are independent of one another (given the model). This clearly does not hold in this dataset as the information about student s_1 is spread over multiple rows, and that about course c_1 as well. Furthermore, tests on $student = s_1$ or $course = c_3$ would be meaningless if we want to learn a model that generalizes to new students.

StarAI approaches take into account the relationships among the individuals as well as deal with uncertainty.

1.3 THE BENEFITS OF MASTERING STARAI

The benefits of combining logical abstraction and relations with probability and statistics are manifold.

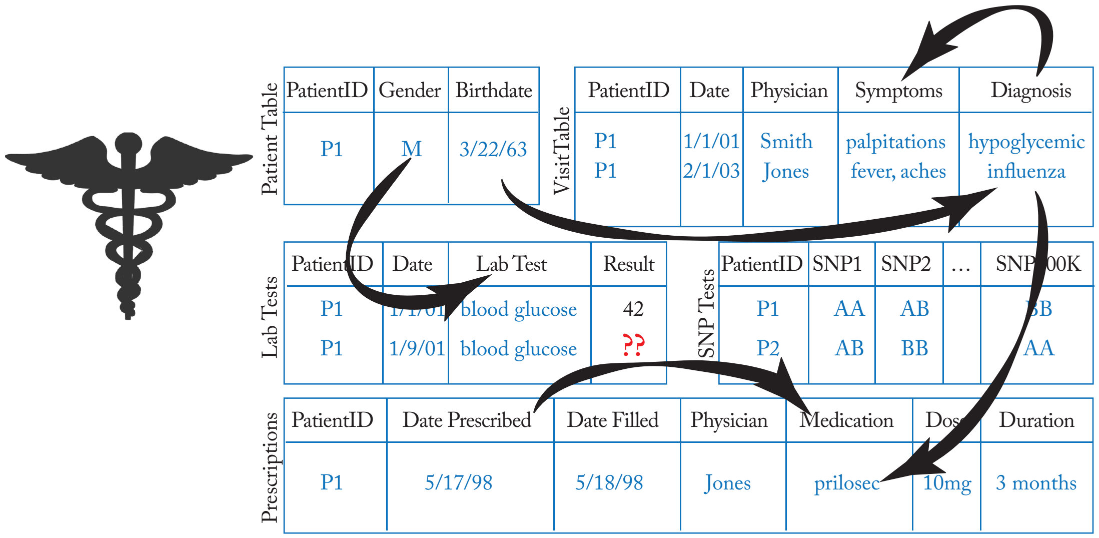
- When learning a model from data, relational and logical abstraction allows one to reuse experience in that *learning about one entity improves the prediction for other entities*. This can generalize to objects that have never been observed before.
- Logical variables, which are placeholders for individuals, allow one to make abstractions that apply to all individuals that have some common properties.
- By using logical variables and unification, one can specify and reason about regularities across different situations using rules and templates rather than having to specify them for each single entity separately.
- The employed and/or learned knowledge is often declarative and compact, which potentially makes it easier for people to understand and validate.
- In many applications, background knowledge about the domain can be represented in terms of probability and/or logic. Background knowledge may improve the quality of learning: the logical aspects may focus the search on the relevant patterns, thus restricting the search space, while the probabilistic components may provide prior knowledge that can help avoid overfitting.

Relational and logical abstraction have the potential to make statistical AI more robust and efficient. Incorporating uncertainty makes relational models more suitable for reasoning about the complexity of the real world. This has been witnessed by a number of real-world applications.

1.4 APPLICATIONS OF STARAI

StarAI has been successfully applied to problems in citation analysis, web mining, natural language processing, robotics, medicine bio- and chemo-informatics, electronic games, and activity recognition, among others. Let us illustrate using a few examples.

Example 1.2 Mining Electronic Health Records (EHRs) As of today, EHRs hold over 50 years of recorded patient information and, with increased adoption and high levels of population coverage, are becoming the focus of public health analyses. Mining EHR data can lead to improved predictions and better disease characterization. For instance, Coronary Heart Disease (CHD) is a major cause of death worldwide. In the U.S., CHD is responsible for approximated 1 in every 6 deaths with a coronary event occurring every 25 s and about 1 death every minute based on data current to 2007. Although a multitude of cardiovascular risks factors have been identified, CHD actually reflects complex interactions of these factors over time. Thus, early detection



PatientID	Gender	Birthdate
P1	M	3/22/63

PatientID	Date	Physician	Symptoms	Diagnosis
P1	1/1/01	Smith	palpitations	hypoglycemic
P1	2/1/03	Jones	fever, aches	influenza

PatientID	Date	Lab Test	Result
P1	1/1/01	blood glucose	42
P1	1/9/01	blood glucose	??

PatientID	SNP1	SNP2	…	SNP500K
P1	AA	AB		BB
P2	AB	BB		AA

PatientID	Date Prescribed	Date Filled	Physician	Medication	Dose	Duration
P1	5/17/98	5/18/98	Jones	prilosec	10mg	3 months

Figure 1.3: Electronic Health Records (EHRs) are relational databases capturing noisy and missing information with probabilistic dependencies (the black arrows) within and across tables.

of risks will help in designing effective treatments targeted at youth in order to prevent cardiovascular events in adulthood and to dramatically reduce the costs associated with cardiovascaular dieases.

Doing so, however, calls for StarAI. As illustrated in Fig. 1.3, EHR data consists of several diverse features (e.g., demographics, psychosocial, family history, dietary habits) that interact with each other in many complex ways making it relational. Moreover, like most data sets from biomedical applications, EHR data contains missing values, i.e., all data are not collected for all individuals. And, EHR data is often collected as part of a longitudinal study, i.e., over many different time periods such as 0, 5, 10 years, etc., making it temporal. Natarajan et al. [2013] demonstrated that StarAI can uncover complex interactions of risk factors from EHRs. The learned probabilistic relational model performed significantly better than traditional non-relational approaches and conforms to some known or hypothesized medical facts. For instance, it is believed that females are less prone to cardiovascular issues than males. The relational model identifies sex as the most important feature. Similarly, in men, it is believed that the low- and high-density lipoprotein cholesterol levels are very predictive, and the relational model confirms this. For instance, the risk interacts with a (low-density lipoprotein) cholesterol level in year 0 (of

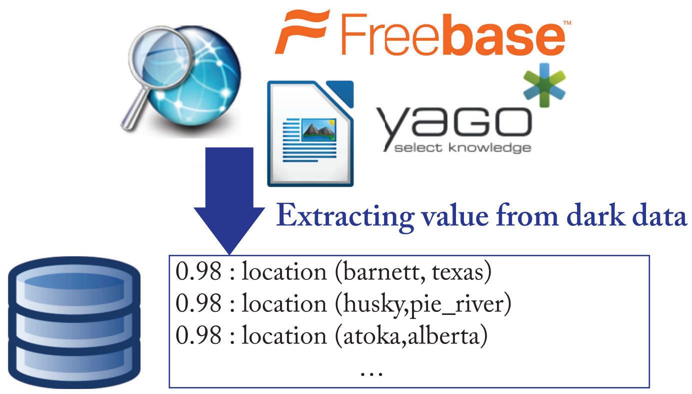

Freebase
yago
select knowledge
Extracting value from dark data
0.98 : location (barnett, texas)
0.98 : location (husky,pie_river)
0.98 : location (atoka,alberta)
...

Figure 1.4: Populating a knowledge base with probabilistic facts (or assertions) extracted from dark data (e.g., text, audio, video, tables, diagrams, etc.) and background knowledge.

the study) for a middle-aged male in year 7, which can result in a relational conditional probability

$$\begin{aligned} 0.79 = P(risk(Person) \mid\ & sex(Person, male) \wedge \\ & (35 \leq age(Person, year\,7) < 45) \wedge \\ & \neg(0 \leq ldl(Person, year\,0) < 100)) \ . \end{aligned}$$

The model also identifies complex interaction between risk factors at different years of the longitudinal study. For instance, smoking in year 5 interacts with cholesterol level in later years in the case of females, and the triglyceride level in year 5 interacts with the cholesterol level in year 7 for males. Finally, using data such as the age of the children, whether the patients owns or rents a home, their employment status, salary range, their food habits, their smoking and alcohol history, etc., revealed striking socio-economic impacts on the health state of the population.

Example 1.3 Extracting value from dark data Many companies' databases include natural language comments buried in tables and spreadsheets. Similarly, there are often tables and figures embedded in documents and web pages. Like dark matter, dark data is this great mass of data buried in text, tables, figures, and images, which lacks structure and so is essentially unprocessable by traditional methods. StarAI helps bring dark data to light (see, e.g., [Niu et al., 2012, Venugopal et al., 2014]), making the knowledge base construction, as illustrated in Fig. 1.4 feasible. The resulting relational probabilistic models are richly structured with many different entity types with complex interactions.

To carry out such a task, one starts with transforming the dark data such as text documents into relational data. For example, one may employ standard NLP tools such as logistic regression, conditional random fields and dependency parsers to decode the structure (e.g., part-of-speech tags and parse trees) of the text or run some pattern matching techniques to identify candidate entity mentions and then store them in a database. Then, every tuple in the database or result of an database query is a random variable in a relational probabilistic model. The next step is to determine which of the individuals (entities) are the same as each other and same as the entities that are already known about. For instance, a mention of an entity may correspond to multiple candidate entities known from some other database such as Freebase. To determine which entity is correct, one may use the heuristic that "if the string of a mention is identical to the canonical name of an entity, then this mention is likely to refer to this entity." In the relational probabilistic model, this may read as the quantified conditional probability statement:

$$\begin{aligned}&\forall Entity, \forall Mention P(entityMentioned(Entity, Mention)\\ &\qquad \mid \exists String\; mentionText(Mention, String) \wedge entityName(Entity, String)) = 0.8\ .\end{aligned}$$

Given such conditional probabilities, the marginal probability of every tuple in the database as well as all derived tuples such as *entityMentioned* can be inferred.

We can also model the relationship between learned features using rich linguistic features for trigger and argument detection and type labeling using weighted logical formulae (defined in Chapter 3) such as:

$$\begin{aligned}0.73 : \;&word(Sentence, PosInSentence, Word) \wedge\\ &triggerType(Sentence, PosInSentence, Type) \wedge\\ &DependencyType(Sentence, PosInDependencyTree, PosInSentence, DepType) \wedge\\ &ArgumentRole(Sentence, PosInDependencyTree, PosInSentence, ArgRole)\ .\end{aligned}$$

For each sentence, this assigns probability to the join of a word at some position in a sentence and the dependency type label that connects the word token to the argument token in the dependency parse tree with the trigger types and argument types of the two tokens. Here, *triggerType* denotes the prediction from a pre-trained support vector machine for triggering some information extracted from the text. This way, high-dimensional, sophisticated features become available to the relational probabilistic model. Moreover, as with Google's Knowledge Vault [Dong et al., 2014], the parameters of the relational probabilistic model or even (parts) of its structure can be learned from data.

Example 1.4 Language modeling The aim of language modeling is to estimate a distribution over words that best represents the text of a corpus. This is central to speech recognition, machine translation, and text generation, among others, and the parameters of language models are commonly used as features or as initialization for other natural language processing approaches. Examples include the word distributions learned by probabilistic topic models, or the word embeddings learned through neural language models. In practice, however, the size of the vocabulary traditionally limited the distributions applicable for this task: specifically, one has to either resort to local optimization methods, such as those used in neural language models, or work with heavily constrained distributions. Jernite et al. [2015] overcame these limitations. They model the entire

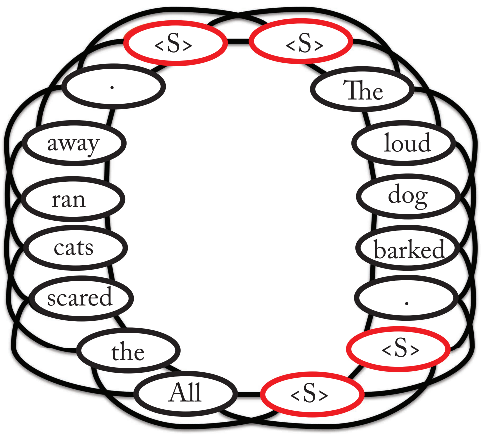


Figure 1.5: Instance of the cyclic language model [Jernite et al., 2015] with added start and end $< S >$ tokens based on two sentences. Each word of a sentence is a random variable and probabilistically depends on the words being at most two steps away. The model exhibits symmetries that are exploitable during parameter estimation using lifted probabilistic inference.

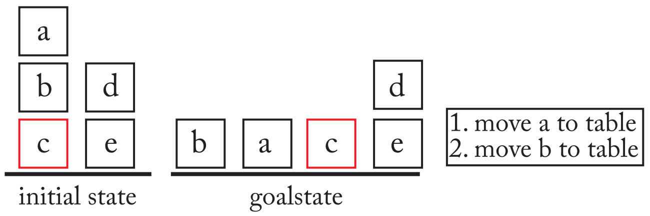


Figure 1.6: A Blocks World planning problem and a plan to solve the goal to have *c* clear.

corpus as an undirected graphical model, whose structure is illustrated in Fig. 1.5. Because of parameter sharing in the model, each of the random variables is indistinguishable and by exploiting this symmetry, Jernite et al. derived an efficient approximation of the partition function using lifted variational inference with complexity independent of the length of the corpus.

Example 1.5 Planning under uncertainty The Blocks World, see Fig. 1.6, consists of a number of blocks stacked into towers on a table large enough to hold them all. The positioning of the towers on the table is irrelevant. The planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower or to the table. However, the effects of moving a block may not be perfectly predictable. For example, the gripper of a robot may be slippery so moving a block may not succeed. This uncertainty compounds the already complex problem of planning a course of action to achieve the goal.

The classical representation and algorithms for planing under uncertainty require the enumeration of the state space. Thus, although arguably a simple planning problem, we get already a rather large representation. For instance, if there are just 10 blocks, there are more than 50 million states. More importantly, we lose the structure implicit in the problem description. Decision-theoretic planning [Boutilier et al., 1999] generalized state-based planning under uncertainty to include features. But features are not enough to exploit all of the structure of the world. Each Blocks World planning instance is composed of these same basic individuals relations, but the instances differ in the arrangements of the blocks. We would like the agent to make use of this structure in terms of computing plans that apply across multiple scenarios. This can be achieved using relational models. For instance, the following parameterized plan, which says that we do nothing if block c is clear. If c is not clear than we move a clear block B that is above c to the floor, achieves the goal to have the block c clear

$$\begin{aligned} \mathit{doNothing} &\text{ if } \mathit{clear}(c) \\ \mathit{move}(B, \mathit{floor}) &\text{ if } \exists\, B\ \mathit{above}(B, c) \wedge \mathit{clear}(B) \wedge \neg\, \mathit{clear}(c)\ , \end{aligned}$$

no matter how many blocks there are.

Example 1.6 Robotics As robots start to perform everyday manipulation tasks, such as cleaning up, setting a table or preparing simple meals, they must become much more knowledgeable than they are today. Typically, everyday tasks are specified vaguely and the robot must therefore infer what are the appropriate actions to take and which are the appropriate individuals involved in order to accomplish these tasks. These inferences can only be done if the robot has access to a compact and general knowledge base [Beetz et al., 2010, Tenorth et al., 2011]. StarAI provides the means for action-centered representation, for the automated acquisition of concepts through observation and experience, for reasoning about and managing uncertainty, and for fast inference.

These are only few examples for the many StarAI applications. It is clear that they all fit the definition of StarAI given above, that is, there are individuals that are connected to one another through relations and there is uncertainty that needs to be taken into account.

There are many more applications of StartAI. For instance, Getoor et al. [2001c] used statistical relational models to estimate the result size of complex database queries. Segal et al. [2001] employed probabilistic relational models for clustering gene expression data and to discover cellular processes from gene expression data [Segal et al., 2003]. Getoor et al. [2004] used probabilistic relational models to understand tuberculosis epidemiology. McGovern et al. [2003] estimated probabilistic relational trees to discover publication patterns in high-energy physics. Neville et al. [2005] used probabilistic relational trees to learn to rank brokers with respect to the probability that they would commit a serious violation of securities regulations in the near future.

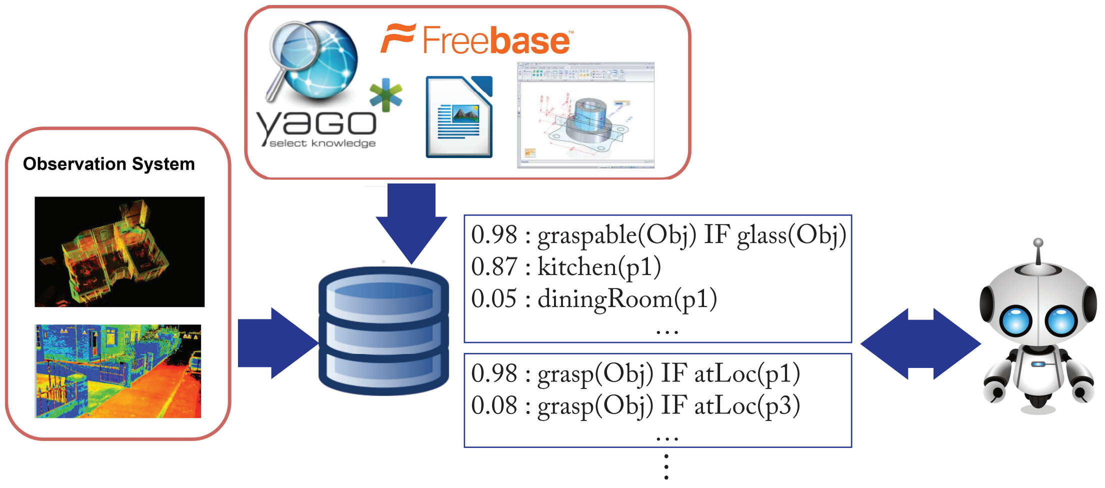

Freebase
yago
select knowledge
Observation System
0.98 : graspable(Obj) IF glass(Obj)
0.87 : kitchen(p1)
0.05 : diningRoom(p1)
...
0.98 : grasp(Obj) IF atLoc(p1)
0.08 : grasp(Obj) IF atLoc(p3)
...

Figure 1.7: The robot's relational probabilistic model is populated with the data produced by its perception and experience (robot log data, human motion tracking, environment information, etc.) as well as with facts (or assertions) extracted from other dark data.

Anguelov et al. [2005] used relational Markov networks for segmentation of 3D scan data. Relational Markov networks have also been used to compactly represent object maps and to estimate trajectories of people by Limketkai et al. [2005]. Kersting et al. [2006] employed relational hidden Markov models for protein fold recognition. Singla and Domingos [2006] proposed a Markov logic model for entity resolution (ER), the problem of finding records corresponding to the same real-world entity. Markov logic has also been used for joint inference for event extraction [Poon and Domingos, 2007, Riedel and McCallum, 2011]. Poon and Domingos [2008] showned how to use Markov logic to perform joint unsupervised coreference resolution. Nonparametric relational models have been used for analyzing social networks [Xu et al., 2009a]. Kersting and Xu [2009] and Xu et al. [2010] used relational Gaussian processes for learning to rank search results. Poon and Domingos [2009] showned how to perform unsupervised semantic parsing using Markov logic networks, and Davis and Domingos [2009] used MLNs to successfully transfer learned knowledge among molecular biology, social network and Web domains. Yoshikawa et al. [2009] used Markov logic for identifying temporal relations in text, Meza-Ruíz and Riedel [2009] for semantic role labeling, and Kiddon and Domingos [2011] for biomolecular event prediction tasks. Niepert et al. [2010] used Markov logic for ontology matching. Tenenbaum et al. [2011] demonstrated that relational models can address some of the deepest questions about the nature and origins of human thought. Verbeke et al. [2012] used statistical relational learning for identifying evidence based medicine categories. Schiegg et al. [2012] segmented motion capture data using Markov logic mixture of Gaussian processes making use of both continuous and declarative observations. Song et al. [2013] presented a Markov logic framework for recognizing complex events from multimodal data. Statistical relational learning has also been employed for learning to predict heart attacks [Weiss et al., 2012], onset of Alzheimer's [Natarajan et al., 2012a], for analyzing pathways and omics experiments [De Maeyer et al., 2013], and extracting adverse drug events from text [Odom et al., 2015]. Lang et al. [2012] applied statistical relational learning to robot manipulations tasks, Nitti et al. [2014] to robot tracking tasks, and Moldovan and De Raedt [2014] to affordance learning. Generally, statistical relational methods are essential for enabling autonomous robots to do the right thing to the right object in the right way [Tenorth and Beetz, 2013]. Hadiji et al. [2015] used relational label propagation to track migration among computer scientists. Kersting et al. [2015] used relational linear programs for topic classification in bibliographic networks, and Toussaint [2015] used relational mathematical programs for combined robot task and motion planning. Foulds et al. [2015] showcased the benefit of relational probabilistic languages for general-purpose topic modeling. Khot et al. [2015b] used Markov logic to answer elementary-level science questions using knowledge extracted automatically from textbooks, and Mallory et al. [2016] used DeepDive to extract both protein–protein and transcription factor interactions from over 100,000 full-text PLOS articles.

It is our belief that there will be many more exciting techniques and applications of StarAI in the future.

1.5 BRIEF HISTORICAL OVERVIEW

Before starting the technical introduction to StarAI, we briefly sketch some key streams of research that have led to the field of StarAI. Note that a full historical overview of this rich research field is out of the scope of this book.

StarAI and SRL (Statistical Relational Learning) basically integrate three key technical ingredients of AI: logic, probability, and learning. While each of these constituents has a rich tradition in AI, there has been a strong interest in pairwise combinations of these techniques since the early 1990s.

- Indeed, various techniques for *probabilistic learning* such as gradient-based methods, the family of EM algorithms or Markov Chain Monte Carlo methods have been developed and exhaustively investigated in different communities, such as in the Uncertainty in AI community for Bayesian networks and in the Computational Linguistics community for Hidden Markov Models. These techniques are not only theoretically sound, they have also resulted in entirely new technologies for, and revolutionary novel products in computer vision, speech recognition, medical diagnostics, troubleshooting systems, etc. Overviews of probabilistic learning can be found in Koller and Friedman [2009], Bishop [2006], Murphy [2012].

- Inductive logic programming and relational learning techniques studied *logic learning*, i.e., learning using first order logical or relational representations. Inductive Logic Programming has significantly broadened the application domain of data mining especially in bio- and chemo-informatics and now represent some of the best-known examples of scientific discovery by AI systems in the literature. Overviews of inductive logic learning and relational learning can be found in this volume and in Muggleton and De Raedt [1994], Lavrac and Dzeroski [1994], De Raedt [2008].

- *Probabilistic logics* have been also studied from a knowledge representational perspective [Nilsson, 1986, Halpern, 1990, Poole, 1993b]. The aim of this research was initially more a probabilistic characterization of logic than suitable representations for learning.

In the late 1990s and early 2000s, researchers working on these pairwise combinations started to realize that they also need the third component. Influential approaches of this period include the Probabilistic Relational Models [Friedman et al., 1999], which extended the probabilistic graphical model approach toward relations, the probabilistic logic programming approach of Sato [1995], which extended the knowledge representation approach of Poole [1993b] with a learning mechanism, and approaches such as Bayesian and stochastic logic programs [Kersting and De Raedt, 2001, Cussens, 2001] working within an inductive logic programming tradition. Around 2000, Lise Getoor and David Jensen organized the first workshop on "Statistical Relational Learning" at AAAI which gathered researchers active in the area for this time. This was the start of a successful series of events that continues till today (since 2010 under the header "StarAI").

In the early days, many additional formalisms were contributed such as RBNs [Jäger, 1997], MMMNs [Taskar et al., 2004], SRMs [Getoor et al., 2001c], MLNs [Richardson and Domingos, 2006], BLOG [Milch et al., 2005], RDNs [Neville and Jensen, 2004], and IBAL [Pfeffer, 2001], many of which are described in the book edited by Getoor and Taskar [2007]. Especially influential was the Markov Logic approach of Richardson and Domingos [2006], as it was an elegant combination of undirected graphical models with relational logic. In this early period, research focused a lot on representational and expressivity issues, often referring to the "alphabet-soup" of systems, reflecting the fact that many systems have names that use three or four letters from the alphabet. Around 2005, it was realized that the commonalities between the different systems are more important than their (often syntactic) differences, and research started to focus on key open issues surrounding learning and inference. One central question is that of lifted inference [Poole, 2003], that is, whether one can perform probabilistic inference without grounding out the probabilistic relational model first. Other questions concern the relationship to existing probabilistic and logical solvers and the continuing quest for efficient inference techniques. Finally, research on SRL and StarAI has also inspired the addition of probabilistic primitives to programming languages, leading to what is now called probabilistic programming. Although some of the early formalisms [Poole, 1993b, Sato, 1995, Pfeffer, 2001] already extend an existing programming language with probabilities, and hence, possess the power of a universal Turing machine, this stream of research became popular since BLOG [Milch et al., 2005] and Church [Goodman et al., 2008].

PART I

Representations

CHAPTER 2
Statistical and Relational AI Representations

Artificial intelligence (**AI**) is the study of computational agents that act intelligently [Russell and Norvig, 2010, Poole and Mackworth, 2010] and, although it has drawn on many research methodologies, AI research arguably builds on two formal foundations: probability and logic.

The basic argument for probability as a foundation of AI is that agents that act under uncertainty are gambling, and probability is the calculus of gambling in that agents who do not use probability will lose to those that do use it (see [Talbott, 2008], for an overview). While there are a number of interpretations of probability, the most suitable for the present book is a Bayesian or subjective view of probability: our agents do not encounter generic events, but have to make decisions in particular circumstances, and only have access to their percepts (observations) and their beliefs. Probability is the calculus of how beliefs are updated based on observations (evidence).

The basic argument for first-order logic is that an agent needs to reason about individuals, properties of individuals, and relations among these individuals, and at least be able to represent conjunctions, implications, and quantification. These requirements arise from designing languages that allow one to easily and compactly represent the necessary knowledge for a non-trivial task. First-order logic has become widely used in the theory of knowledge representation. It assumes that the world consists of individuals, among which various relations hold or do not hold. Logic provides a semantics, a specification of meaning, which agent designers can then choose to implement in various ways.

Both probability and predicate calculus can be seen as extending propositional logic, one by adding measures over the worlds, and the other by extending the propositional calculus to include individuals and relations. Since, statistical relational AI (StarAI) aims to seamlessly combine these two strands of research, we will first introduce each of these separately before moving on to presenting their possible integrations.

2.1 PROBABILISTIC GRAPHICAL MODELS

The semantics of probability [Halpern, 1990, 2003] can be defined in terms of **possible worlds** and **random variables** (although they are neither random nor variable). We can either define random variables in terms of possible worlds (a random variable is a function on possible worlds) or define possible worlds in terms of random variables (a possible world is an assignment of a value to every random variable). Either way, a random variable has a value in every world. A random variable having a particular value is a proposition. Probability is defined in terms of a non-negative measure over sets of possible worlds that follow some intuitive axioms (see, e.g., [Halpern, 2003]). This means that a probabilistic graphical model defines a probability distribution over its possible worlds, or equivalently a joint probability distribution over the propositions, or the assignments of values to the random variables.

In Bayesian probability, we make explicit assumptions and the conclusions are consequences of the specified knowledge and assumptions. The assumptions are about the space of possible hypotheses and the prior probabilities. One particular explicit assumption is the assumption of conditional independence. **graphical models** [Pearl, 1988, Lauritzen, 1996, Koller and Friedman, 2009] provide representations of such conditional independence of random variables in terms of graphs. A **Bayesian network** [Pearl, 1988, Darwiche, 2009] is an acyclic directed graphical model of probabilistic dependence where the nodes are random variables. A Bayesian network encapsulates a directed form of independence between the random variables in the graph: a variable is conditionally independent of its non-descendants given its parents. This has turned out to be a very useful assumption in practice. A Bayesian network requires a representation of the conditional probability of each variable given its parents.

Undirected graphical models, or **Markov networks**, [Pearl, 1988] are defined in terms of factors (or potentials) among random variables. A **factor** represents a function of a set of random variables; for example, a factor on variables $\{X, Y, Z\}$ is a function that given a value for each variable, returns a non-negative number. The nodes in a Markov network correspond to random variables, and the variables in a factor are neighbors of each other in the graph. These models encapsulate the assumption that a variable is independent of other variables given all of its neighbors. The next two sections give more details.

2.1.1 BAYESIAN NETWORKS

Formally, a **Bayesian network** (or **belief network**) is an augmented, acyclic directed graph (DAG), where each node corresponds to a random variable X_i and each edge indicates a **direct influence** among the random variables. The influence for each variable X_i is quantified with a conditional probability distribution (CPD) $\mathbf{P}(X_i \mid \mathbf{Pa}(X_i))$, where $\mathbf{Pa}(X_i)$ are the parents of X_i in the graph. The Bayesian network represents a set of independence assumptions: a variable is independent of the other variables that are not its descendents given its parents.

Example 2.1 As an example of a Bayesian network, consider Judea Pearl's famous burglary alarm network. The Bayesian network in Fig. 2.1 contains the random variables *burglary*, *earthquake*, *mary_calls*, *john_calls*, and *alarm*. The network specifies that *burglary* and *earthquake* are independent, and that *john_calls* is independent of the other variables given *alarm*. Assume that the random variables are all Boolean, i.e., they can have the range $\{true, false\}$, then the Bayesian network defines a probability distribution over truth-assignments to $\{alarm, earthquake, mary\_calls,$

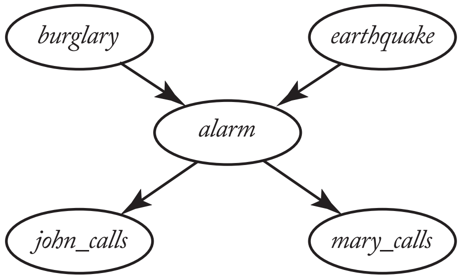


Figure 2.1: A Bayesian network for burglary alarms (adapted from Judea Pearl). Nodes denote random variables and edges denote direct influences among the random variables.

Table 2.1: Conditional probability distributions associated with the nodes in the burglary alarm network, cf. Fig. 2.1

$\mathbf{P}(burglary)$
$(0.001, 0.999)$

$\mathbf{P}(earthquake)$
$(0.002, 0.998)$

burglary	*earthquake*	$\mathbf{P}(alarm \mid burglary, earthquake)$
true	*true*	$(0.95, 0.05)$
true	*false*	$(0.95, 0.05)$
false	*true*	$(0.29, 0.71)$
false	*false*	$(0.001, 0.999)$

alarm	$\mathbf{P}(john\_calls \mid alarm)$
true	$(0.90, 0.10)$
false	$(0.05, 0.95)$

alarm	$\mathbf{P}(mary\_calls \mid alarm)$
true	$(0.70, 0.30)$
false	$(0.01, 0.99)$

john_calls, *burglary*}. To do so, it makes use of CPDs associated to each of the nodes are specified in Table 2.1. They include the CPDs $\mathbf{P}(alarm \mid burglary, earthquake)$, $\mathbf{P}(earthquake)$, etc.

The Bayesian network thus has two components: a qualitative one—the directed acyclic graph—and a quantitative one—the conditional probability distributions. Together they specify the joint probability distribution.

Because of the conditional independence assumption, we can write down the joint probability density as

$$\mathbf{P}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i \mid \mathbf{Pa}(X_i)) \ . \tag{2.1}$$

The independence assumption, which states that each variable is independent of its non-descendents given its parents, implies other independencies. These are axiomatized by d-separation [Pearl, 1988], which allows to infer all independencies that are implied from the independence assumptions of Bayesian networks.

2.1.2 MARKOV NETWORKS AND FACTOR GRAPHS

Conceptually, a **Markov random field** [Pearl, 1988] is an undirected analogue of a Bayesian network. A Markov network is defined in terms of a set of discrete-valued random variables, $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ and a set of factors $\{f_1, \ldots, f_m\}$, where a factor is a non-negative function of a subset of the variables.

A Markov network represents the joint distribution over $\mathbf{X}$ as proportional to the product of the factors, i.e.,

$$\begin{aligned} P(\mathbf{X} = \mathbf{x}) &= \frac{1}{Z} \prod_k f_k(\mathbf{X}_k = \mathbf{x}_k) \\ Z &= \sum_{\mathbf{x}} \prod_k f_k(\mathbf{X}_k = \mathbf{x}_k) \ , \end{aligned}$$

where $f_k(\mathbf{X}_k)$ is a factor on $\mathbf{X}_k \subseteq \mathbf{X}$, and $\mathbf{x}_k$ is $\mathbf{x}$ projected onto $\mathbf{X}_k$. Z is a normalization constant known as the **partition function**.

As long as the factors are all non-zero (they always return a positive number), the distribution can be represented as a **log-linear model**:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left[\sum_k w_k \cdot g_k(\mathbf{x})\right], \tag{2.2}$$

where the factors $g_k(\cdot)$ are functions of (a subset of) the configuration $\mathbf{x}$, and w_i are real-valued weights.

A **Markov network** is a graphical representation of a Markov random field where the nodes are the random variables and there is an arc between any two variables that are in a factor together.

As for Bayesian networks, the structure of a Markov network encodes conditional independencies. Basically, two nodes X and Y in a network are conditionally independent given variables $Z_1, ldots, Z_n$ (i.e., $P(X \mid Y, Z_1, \ldots, Z_n) = P(X \mid Z_1, \ldots, Z_n)$) if and only if all paths from X to Y contain one of the variables Z_i. The Markov network for the alarm example is illustrated in Fig. 2.2.

A Bayesian network can be seen as a Markov random field, where each conditional probability is a factor. Thus any inference algorithm developed for Markov networks can be applied to Bayesian networks. Many of the inference algorithms are designed for Markov networks for this reason. However, Bayesian networks allow for specialized algorithms, such as recursively pruning variables that are not observed, not queried, and have no children. Any Markov network can also be represented as a Bayesian network by scaling each factor to be in the range [0, 1] (by multiplying by a constant), creating a new variable t_k for each factor f_k and defining $P(t_k = true \mid \mathbf{x}_k) = f_k(\mathbf{x}_k)$, and conditioning on $t_k = true$.

An alternative way to depict a Markov random field is in terms of a **factor graph**. A factor graph is a bipartite graph, which contains a variable node for each random variable x_i and a factor node for each factor f_i. There is an edge between a variable node and a factor node if and only if the

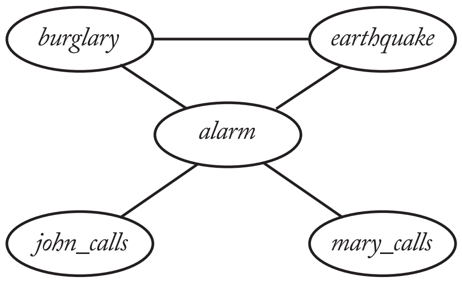


Figure 2.2: The Markov network for the burglary alarm Network.

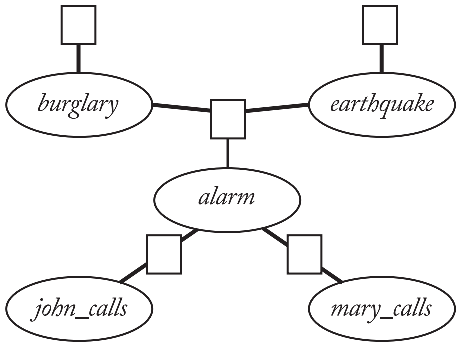


Figure 2.3: The factor graph representing the burglary alarm Bayesian network.

variable appears in the factor. This is often the preferred representation to describe probabilistic inference methods sending messages between variables and factors. The factor graph for the *alarm* Bayesian network is shown in Fig. 2.3.

A Markov network can be obtained from a factor graph, by adding arcs between all of the neighbors of each factor node, and dropping the factor nodes. A factor graph can be obtained from a Markov network by creating a factor for each clique in the Markov network. However, mapping to a Markov network loses information. For example, the Markov random field with a single factor $f_0(X, Y, Z)$ has the same Markov network as the Markov random field with factors $\{f_1(X, Y), f_2(Y, Z), f_3(X, Z)\}$, but they have different factor graphs. The network with a single factor can represent more distributions than the one with only pairwise factors, e.g., the distribution where the variables are independent of each other except that they have even parity.

```
parent(jef, paul).
parent(paul, ann).
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

Figure 2.4: A logic program defining *grandparent*.

```
nat(0).
nat(s(X)) :- nat(X).
```

Figure 2.5: A logic program defining *nat*.

2.2 FIRST-ORDER LOGIC AND LOGIC PROGRAMMING

The probabilistic models introduced in the previous section have a fixed finite number of random variables. Boolean random variables correspond to propositions in logic, and propositions can also represent more complex random variables. First-order logic allows for an unbounded number of propositions by including logical variables, constants, and functions. The same ideas can be used for random variables. In order to understand this we now introduce first-order logic, and the directed variant, which are logic programs.

To introduce logic and logic programs, consider Figs. 2.4 and 2.5, which contain two programs, *grandparent* and *nat*. In these programs *grandparent*/2, *parent*/2, and *nat*/1 are **predicates** (with their *arity*, the number of arguments, listed explicitly). The strings *jef*, *paul*, *ann*, and 0 are **constants**, whereas X, Y, and Z are **logical variables**, which we will just call **variables** when there is no confusion with random variables. All constants and variables are also **terms**. In addition, there exist structured terms such as $s(X)$, which contains the **functor** $s/1$ of arity 1 and the term X. Constants are often considered as functors of arity 0. A first-order **alphabet** Σ is a set of predicate symbols, constant symbols and functor symbols.

Atoms are predicate symbols followed by the necessary number of terms, e.g., *parent*(*jef*, *paul*), *nat*($s(X)$), *parent*(X, Z), etc. A **literal** is an atom, e.g., *nat*($s(X)$), (called a positive literal) or the negation of an atom, e.g., $\neg$*nat*($s(X)$) (called a negative literal).

First-order logical **formulas** are recursively constructed from atoms using **logical connectives** ($\neg$, $\vee$ and $\wedge$) and **quantifiers** ($\forall$ and $\exists$). If f_1 and f_2 are formulas then the following are formulas, too:

- $\neg f_1$ (negation), which is true iff f_1 is not true;
- $f_1 \wedge f_2$ (conjunction), which is true iff both f_1 and f_2 are true;
- $f_1 \vee f_2$ (disjunction), which is true iff either f_1 or f_2 is true;

- $f_1 \rightarrow f_2$, or $f_2 \leftarrow f_1$ (implication), which is true if f_1 is false or f_2 is true;
- $f_1 \leftrightarrow f_2$ (equivalence), which is shorthand for $(f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$;
- $\forall X f_1$ (universal quantification), which is true if f_1 is true for every individual that X could refer to; and
- $\exists X f_1$ (existential quantification), which is true if f_1 is true for at least one individual.

An important subset of first-order logic is that of **clausal logic**. A **clausal theory** consists of a set of clauses. A **clause** is a formula of the form

$$\underbrace{A_1 \vee \ldots \vee A_n}_{\text{conclusion}} \underbrace{\leftarrow}_{\text{if}} \underbrace{B_1 \wedge \ldots \wedge B_m}_{\text{body}}, \tag{2.3}$$

where the A_j and the B_i are atoms. All variables in clauses are understood to be universally quantified. Clause (2.3) is logically equivalent to

$$A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \cdots \vee \neg B_m \ .$$

Example 2.2 Two example clauses are:

$$\begin{aligned} female(X) \vee male(X) &\leftarrow human(X) \\ false &\leftarrow female(X) \wedge male(X) \ . \end{aligned}$$

The first clause states that for all X if X is *human*, X is either *male* or *female*. The second clause states that for all X, X cannot be both *male* and *female*, that is, no-one is both male and female. Here, *false* is an atom that is always false or is the empty disjunction $n = 0$.

A practical subset of clausal logic is that of logic programs, which are a subset that contains no uncertainty; what is given up is the ability to state that $a \vee b$ is true, without saying which one of a or b is true. Logic programs are of interest because they have an intuitive declarative and procedural interpretation [Kowalski, 2014].

A **definite clause** is a clause with exactly one atom in the conclusion part of the clauses (that is, $n = 1$). Following the Prolog tradition, definite clauses are usually written with the conjunction represented by a comma, and "if" represented by ":-"

$$A \text{:-} B_1, \ldots, B_m \ .$$

A is the **head** of the clause, and $B_1, \ldots, B_m$ is the **body** of the clause. If $m = 0$ the implication ":-" can be omitted and the clause is called a **fact**. A **logic program** is a set of definite clauses.

Example 2.3 The definite clause

$$grandparent(X, Y) \text{:-} parent(X, Z), parent(Z, Y)$$

can be read as: for all X, for all Y and for all Z: X is a *grandparent* of Y if X is a *parent* of Z and Z is a *parent* of Y. $grandparent(X, Y)$ in the head of the clause, and $parent(X, Z), parent(Z, Y)$ is the body of the clause.

Example 2.4 Figure 2.4 shows a logic program defining *grandparent*/2 with two *parent*/2 facts and Fig. 2.5 a program defining *nat*/1.

Reasoning with logic involves manipulating the formula as data structures. An **expression** is a term, atom, conjunction, or clause. The set of logical variables in expression E is denoted as $\mathrm{Var}(E)$. For example, in the clause c defined in Example 2.3, $\mathrm{Var}(c) = \{X, Y, Z\}$. An expression E is **ground** when there is no variable occurring in E, i.e., $\mathrm{Var}(E) = \emptyset$. To connect variables and (structure) ground terms, we make use of substitutions.

A **substitution** is of the form $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$, e.g., $\{Y/ann\}$, where the V_i are distinct logical variables, and t_i are terms. Applying a substitution θ to an expression e yields the expression $e\theta$ where all occurrences of the variables V_i are simultaneously replaced by the term t_i, e.g., if c is the definite clause (2.3) above, $c\{Y/ann\}$ is

$$grandparent(X, ann) :\!- parent(X, Z), parent(Z, ann)\ .$$

A substitution θ is called a **unifier** for two expressions e_1 and e_2 if $e_1\theta = e_2\theta$, that is, if the expressions become identical after applying the substitution. For instance, the substitution $\{X/jef, Y/ann\}$ is a unifier for $parent(X, ann)$ and $parent(jef, Y)$.

It may seem that it is impossible to reason about clauses where variables can represent anything, e.g., numbers, people, or symbols. It turns out that if a set of clauses has a model where the variables can represent anything, it has a model where the variables denote symbolic objects in the Herbrand base. The **Herbrand base** of a set of clauses P, denoted as $\mathrm{hb}(P)$, is the set of all ground atoms constructed with the predicate, constant and function symbols in the alphabet of P (and inventing a new constant if P doesn't contain any constants).

Example 2.5 The Herbrand bases of the *grandparent* and *nat* logic programs in Figs. 2.4 and 2.5 are

$$\begin{aligned}\mathrm{hb}(grandparent) = \{&parent(ann, ann), parent(jef, jef),\\ &parent(paul, paul), parent(ann, jef), parent(jef, ann), \ldots,\\ &grandparent(ann, ann), grandparent(jef, jef), \ldots\},\\ \mathrm{hb}(nat) = \{&nat(0), nat(s(0)), nat(s(s(0))), \ldots\}\ .\end{aligned}$$

$\mathrm{hb}(grandparent)$ has 18 elements and $\mathrm{hb}(nat)$ is infinite.

When considering the constants a and b, the clausal theory for Example 2.2 has the following Herbrand base:

$$\mathrm{hb}(human) = \{human(a), human(b), female(a), female(b), male(a), male(b)\}\ .$$

A **Herbrand interpretation** for a set of clauses P is a subset of $\mathrm{hb}(P)$. Herbrand interpretation I represents the possible world where all of the elements in I are true and the elements of $\mathrm{hb}(P) \setminus I$ are false. A Herbrand interpretation I is a **model** of clause c, if and only if, for all substitutions, θ that grounds all variables in c such that $\mathrm{body}(c)\theta \subseteq I$ holds, it also holds that $\mathrm{head}(c)\theta \in I$. Interpretation I is a model of a set of clauses P if I is a model of all clauses in P. A clausal theory P **entails** a clausal theory P', denoted as $P \models P'$, if and only if, each model of P is also a model of P'.

Example 2.6 The following two interpretations are models for Example 2.2, which shows that a set of clauses can have more than one model:

$$\{human(an), female(an), human(paul), male(paul)\}$$
$$\{human(an), male(an), human(paul), female(paul)\}\ .$$

These models are also minimal w.r.t. set inclusion.

While clausal theories may have no, one, or more minimal models, a definite clause program has a unique **least Herbrand model**. The least Herbrand model $\mathrm{LH}(P)$, consists of all ground atoms $f \in \mathrm{hb}(P)$ such that P logically entails f, i.e., $P \models f$. It is also a *minimal* model. All ground atoms in the least Herbrand model are provable.

Example 2.7 The least Herbrand models of our programs from Example 2.5 are

$$\begin{aligned} \mathrm{LH}(grandparent) &= \{parent(jef, paul), parent(paul, ann), grandparent(jef, ann)\} \\ \mathrm{LH}(nat) &= hb(nat)\ . \end{aligned}$$

That is, $nat(a)$ is provable for all atoms a in the Herbrand base of nat.

In Part II on inference, we will discuss how to use clausal logic for inference and for reasoning. This will also include the computation of the least Herbrand model of a logic program. For a detailed introduction to logic programming, see Lloyd [1989], for a more gentle introduction, see Flach [1994], and for a detailed discussion of Prolog, see Sterling and Shapiro [1986].

CHAPTER 3
Relational Probabilistic Representations

Probability theory can be seen as extending the propositional calculus to include uncertainty; we can ask for the probability of a proposition conditioned on a proposition. Likewise, the (first-order) predicate calculus can be seen as extending propositional calculus to include relations (represented by predicate symbols), individuals, and logical variables that can be universally or existentially quantified. Relational probabilistic representations can be seen as extending the predicate calculus to include probabilities, or extending probabilistic models to include relations, individuals, and variables.

Probability has seen a resurgence in AI due mainly to exploiting the independency inherent in graphical models, and most relational probabilistic models are built from these graphical models. In **relational probabilistic models**, the random variables are built from individuals and relationships among individuals. Just as the first-order predicate calculus extends the propositional calculus to reason about individuals and relations, relational probabilistic models extend probabilistic models, such as Bayesian networks and Markov networks, to allow reasoning about individuals and relations.

With such models, a reasoning agent observes individuals, their properties, and relations among them, and conditioning on these allows probabilistic predictions about individuals, properties and relationships. We often want to build the models before we know which individuals exist in a population so that the models can be applied to diverse populations. When we condition on observations of particular individuals and relations among particular individuals, we can learn something about those individuals, which can then be used for subsequent reasoning and learning. In the open-universe case, we can be unsure about whether two descriptions or names refer to the same individual, whether an individual that fits a description actually exists or how many individuals there are.

These are key properties of StarAI domains that we have encountered already in the examples used in the introduction.

Example 3.1 Education modeling In an educational domain (see, e.g., Fig. 1.2), the individuals may be students, courses, tests, or particular questions on tests. We want to condition on the observations of what tests the students took, what answers they provided, and any other pertinent information to predict what they understand.

Example 3.2 Medical diagnosis In a medical diagnosis system (see, e.g., Fig. 1.3), the individuals may be patients, treatments, drugs, tests, particular lumps on a person's body, etc. A system can condition on a patient's electronic health record to predict the outcome of some treatment. Having a relational model is sensible since electronic health records are not summaries, but contain a history of physical observations, treatments, tests, etc. They differ greatly in the amount of detail they contain per person.

In both of these examples the models are not about a particular patient or student, but are models that can be applied to any patient or student. The set of observations is not fixed, as for instance an electronic health record may contain an unbounded number of observations and test results about multiple lumps that may have appeared (and disappeared) on a patient.

The main property relational models exploited is that of **exchangeability**: those individuals about which we have the same information should be treated identically. This is the idea behind (universally quantified) variables; some statements are true for all individuals. In terms of probabilistic models, this means we can exchange the constants in any grounding, and still get the same probabilities for any proposition. This implies that before we know anything particular about any of the individuals, they all should share their probabilistic parameters. It also provides a form of symmetry that can be exploited for representations, inference and learning. De Finetti [1974] shows how distributions on exchangeable random variables can be represented in terms of directed models with latent variables. Note that exchangeability of individuals corresponds to exchangeability of random variables only for properties (relations on single individuals), but not for properties of multiple individuals (but see [Kallenberg, 2005] for results in such situations).

Over the years, a multitude of different languages and formalisms for probabilistic relational modeling have been devised. However, there are also some general design principles underlying this "alphabet soup," which we will discuss first.

3.1 A GENERAL VIEW: PARAMETERIZED PROBABILISTIC MODELS

Relational probabilistic models are typically defined in terms of **parameterized random variables** [Poole, 2003], which are often drawn in terms of **plates** [Buntine, 1994, Jordan, 2010]. A parameterized random variable corresponds to a predicate or a function symbol in logic.

We use a first-order alphabet consisting of **logical variables**, **constants**, **function symbols**, and **predicate symbols**. We assume that the logical variables are typed, where the domain of the type (i.e., the set of individuals of the type), is called the **population**. Recall from Section 2.2 that a **term** is either a logical variable, a constant or of the form $f(t_1, \ldots, t_k)$, where f is a **function symbol** and each t_i is a term. We treat constants as function symbols with no arguments (and so are not treated specially). Each function has a range, which is a set of values. In the following we treat relations as Boolean functions (with range $\{true, false\}$).

A **parameterized random variable** (PRV) is of the form $f(t_1, \ldots, t_k)$ where each t_i is a term and f is a function (or predicate) symbol. A **random variable** is a parameterized random variable which does not contain a logical variable. An **atom** is of the form $f(t_1, \ldots, t_k) = v$ where v is in the range of f. When the range of f is $\{True, False\}$, (i.e., when f is a predicate symbol), we write $f(t_1, \ldots, t_k) = True$ as $f(t_1, \ldots, t_k)$, and $f(t_1, \ldots, t_k) = False$ as $\neg f(t_1, \ldots, t_k)$. We can build a formula from relations using the standard logical connectives. The **grounding** of a PRV is the set of random variables obtained by uniformly replacing each logical variable by each individual in the population corresponding to the type of the logical variable.

A **lifted graphical model**, also-called a **template-based model** [Koller and Friedman, 2009], is a graphical model with parameterized random variables as nodes, and a set of factors among the parameterized random variables, called **parameterized factor**. A lifted model, together with an assignment of a population to each logical variable means its grounding: the graphical model where the set of random variables is the set of groundings of the PRVs in the model, and the factor in the ground model is the grounding of the corresponding factor of the lifted model. The details differ in the various representation languages, because what is allowed as factors varies, but the issues can be highlighted by a few examples.

Example 3.3 Consider a model of the performance of students in courses. With such a model, we could, for example, condition on the data of Fig. 1.2 to predict how students s_3 and s_4 will perform in course c_4. Suppose we have the types: *student* and *course*. The population of *student* is the set of all students, and the population of *course* is the set of all courses. The parameterized random variable $grade(S, C)$ could denote the grade of student S in course C. For a particular student *sam*, and course *cs*101, the instance $grade(sam, cs101)$ is a random variable denoting the grade of Sam in the course *cs*101. The range of *grade* could be the set of possible grades, say $\{a, b, c, d, f\}$. Similarly, $int(S)$ could be a parameterized random variable denoting the intelligence of student S. The PRV $diff(C)$ could represent the difficulty of course c.

If there are n students and m courses, the grounding contains nm random variables that are instances of $grade(S, C)$, n instances of $int(S)$ and m instance of $diff(C)$. Thus, there are $nm + n + m$ random variables in the grounding.

Figure 3.1 gives a plate representation of a directed model to predict the grades of students in courses. In this figure, s is a logical variable that denotes a student and c is a logical variable that denotes a course. Note that this figure redundantly includes the logical variables in the plates as well as arguments to the parameterized random variables. Such parameterized models represent their grounding, where there is an instance of each random variable for each assignment of an individual to a logical variable. The factors of the ground network are the groundings of the corresponding factors of the relational model. Figure 3.2 shows such a grounding where there are three students Sam, Chris, and Kim and two courses ($c1$ and $c2$).

Note the **conditional independence** assumptions in this example (derived from the independence inherent in the underlying Bayesian network): the intelligence of the students are independent of each other given no observations. The difficulty of the courses are independent of each other, given no observations. The grades are independent given the intelligence and the difficulty. Given no observations, a pair of grades that share a student or a course are dependent

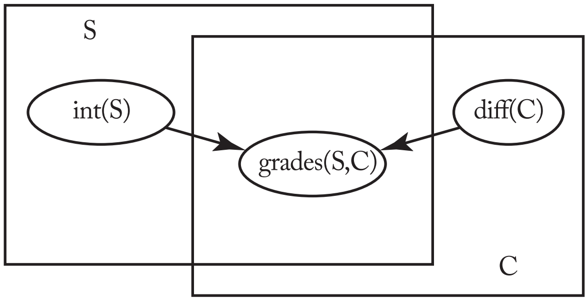


Figure 3.1: Plate representation of the grades model.

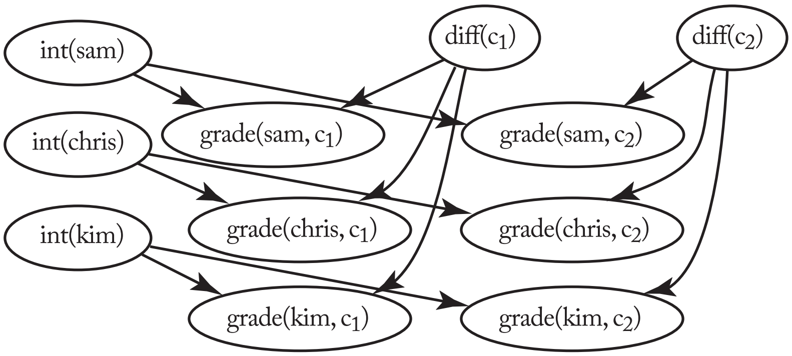


Figure 3.2: Grounding of the grades model for 3 people (Sam, Chris, and Kim) and 2 courses (c_1 and c_2).

on each other, as they have a common parent, but a pair of grades about different students and courses are independent. Given observations on grades, the intelligence variables and the difficulty variables can become interdependent.

The basic principle used by these models is that of **parameter sharing**: the instances of the parameterized factors created by substituting constants for logical variables share the same probabilistic parameters. This is a consequence of exchangeability: the variables can denote any individual, and so need to be the same for each instance (before we have observed anything to distinguish the individuals). The various languages differ in how to specify the conditional probabilities of the parameterized random variables given their parents, or the other parameters of the probabilistic model.

Often in plate models [Buntine, 1994, Jordan, 2010] the numerical parameters are made explicit, to emphasize that the probabilistic parameters do not depend on the individuals.

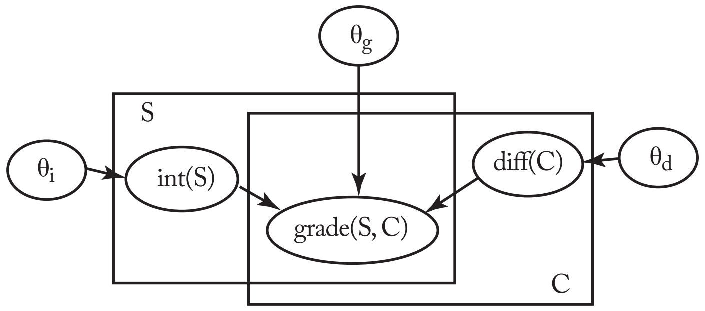


Figure 3.3: Plate representation of the grades model, with shared parameters explicit.

Example 3.4 Figure 3.3 shows the plate model of Fig. 3.1 with the numerical parameters made explicit. Parameter θ_i specifies the prior probability of *int*(S) that is shared among the students, parameter θ_d specifies the prior probability of *diff*(C), and θ_g specifies the numerical parameters of the conditional probability $P(\mathit{grade}(S, C) \mid \mathit{int}(S), \mathit{diff}(C))$. The figure makes explicit that these parameters do not depend on the individuals.

There are some issues that relational models must face, beyond the non-relational case. We highlight one such issue now.

Suppose there is a directed model where for some parameterized random variable X, the parents of X include a logical variable that does not appear in X. In the grounding, X has an unbounded number of parents, and we need some way to represent the conditional probability of a variable given an unbounded number of parents; such methods are called **aggregation functions**. While aggregation does occur in the non-relational case, it cannot be avoided in the directed relational case.

Example 3.5 Consider the problem of determining guilt in a shooting. Figure 3.4 depicts an example where someone shot person Y if there exists a person X who shot Y. In this model, *someone_shot*(Y) has the number of parents equal to the population size. In this case, the appropriate aggregator is a logical "or." The PRV *someone_shot*(Y) has the meaning $\exists X\, \mathit{shot}(X, Y)$, and it may be appropriate to write it in this way. Whether X shot Y depends on whether X had motive, opportunity and a gun.

The instances of *shot*(X, Y) for each X are dependent because they share a common ancestor, *has_gun*(X). The instances of *shot*(X, Y) are dependent for some Y if *someone_shot*(Y) is observed.

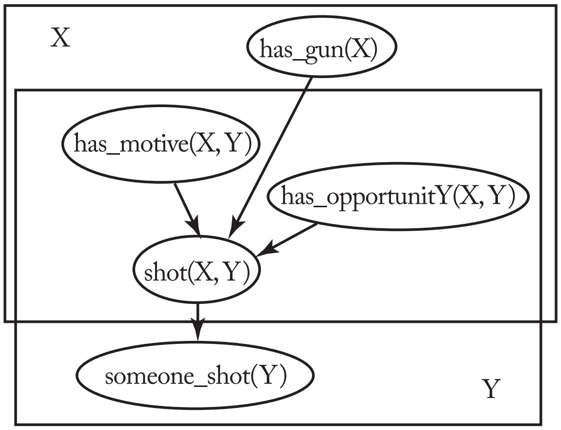


Figure 3.4: Plate representation of determining guilt from Example 3.5.

Often there is much more structure about how the instances interact than can be represented in a model that assumes that the instances are only dependent given shared ancestors or observed shared descendants, as in the following two examples.

Example 3.6 Consider the case of diagnosing students' performance in adding multi-digit numbers of the form

$$\begin{array}{rrrr} & & x_1 & x_0 \\ + & & y_1 & y_0 \\ \hline & z_2 & z_1 & z_0 \end{array}.$$

A student, given values for the digits x_i and y_i, provides values for the z_i digits. The aim is to determine whether the students have mastered addition from observations of their performance.

Whether a student gets the correct carry depends on the previous x, y and carry, and whether the student knows how to carry. This dependency can be seen in Fig. 3.5. Here $x(D, P)$ is a parameterized random variable representing digit D of the first summand for problem P. There is a random variable in the grounding for each digit D and each problem P. A ground instance, such as $x(3, problem_{57})$, is a random variable that may represent the third digit from the right of the topmost number of problem 57. Similarly, there is a z-variable for each digit D, problem P, student S, and time T. The plate notation can be read as duplicating the random variable for each tuple of individual the plate is parameterized by. Whether the student knows how to carry or knows how to add depends on the student and the time, but not on the digit or the problem.

For each problem, student and time, the carry for digit D, namely $c(D, P, S, T)$, depends, in part, on $c(D-1, P, S, T)$, that is, on the carry from the previous digit. There is a special case for the first digit. Similarly, whether a student knows how to add or carry at any time depends on whether they knew how to add or carry at the previous time. This is depicted by cycles in the

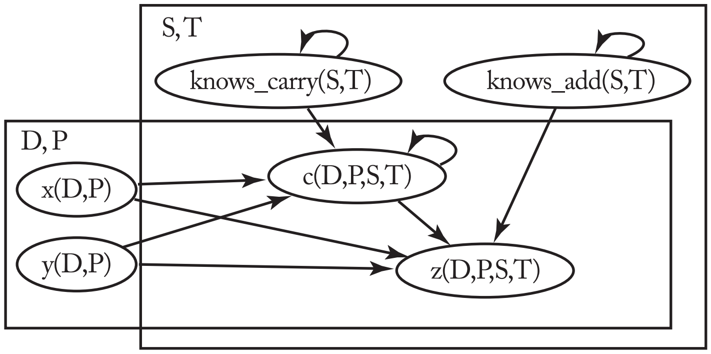


Figure 3.5: Belief network with plates for multidigit addition.

plate notation, but it results in an acyclic grounding. The plate notation does not convey all of the information about the dependencies.

Example 3.7 Consider determining the probability that two authors are collaborators, which may depend on whether they have written papers in common, or even whether they have written papers apart from each other. That is, $collaborates(A_1, A_2)$ may depend on the existence of, and properties of any paper P such that $author(A_1, P) \wedge author(A_2, P)$ is true. A representation has to be able to specify what happens when there is more than one paper that they are co-authors of. The existential quantification used in Example 3.5 does not seem to be appropriate here. It may also depend on other co-authors who may have collaborated with each of them. Any such rules are only applicable if $A_1 \neq A_2$, that is if A_1 and A_2 are different people. Examples such as these require more sophisticated languages than the plate models specified above.

The first such languages either had explicit languages for constructing the ground network [Breese, 1992, Goldman and Charniak, 1990] or defined templates for prior and conditional probabilities [Horsch and Poole, 1990] directly in terms of tables, and required a combination function (such as noisy-and or noisy-or) when the number of parents depends on the number of individuals. These template models turn out not to be very flexible as they cannot represent the subtleties involved in how one random variable can depend on others.

To represent such examples, it is useful to be able to specify how the logical variables interact, as is done in logic programs. The independent choice logic (ICL) [Poole, 1997, 2008] (originally called probabilistic Horn abduction (PHA) [Poole, 1991, 1993b]) allows for arbitrary (acyclic) logic programs (including negation as failure) to be used to represent the dependency. The conditional probability parameters are represented as independent probabilistic inputs to the logic program. A logic program that represents Example 3.6 is given in Chapter 14 of [Poole and Mackworth, 2010]. This mix of probabilities and logic programs also forms the foundations for PRISM [Sato and Kameya, 1997, 2008], which has concentrated on learning, and for Problog [De Raedt et al., 2007], a project to build an efficient and flexible language.

Other languages are based on entity-relationship models [Friedman et al., 1999, Heckerman et al., 2004], fragments of first-order logic [Muggleton, 1996, Kersting and De Raedt, 2007, Laskey, 2008], or even in terms of programming languages such as in IBAL [Pfeffer, 2007] or Church [Goodman et al., 2008]. Undirected models, exemplified by Markov logic networks (MLNs) [Richardson and Domingos, 2006], have a similar notion of parameterized random variables, but the probabilities are represented as weights of first-order clauses. The meaning of these relational models is represented by their grounding, but for MLNs the grounding is a Markov network rather than a Bayesian network (see Pearl [1988] for a discussion of the relationship between Bayesian and Markov networks). Yet other languages are extensions of the maximum entropy principle to the relational setting [Kern-Isberner and Thimm, 2010, Beierle et al., 2015] or incorporate probabilistic reasoning about similarities and relational structures [Bröcheler et al., 2010, Bach et al., 2013].

All of the representations are required to be finite (for otherwise they cannot be written down), but the grounding does not need to be finite. The logic-programming based languages such as ICL and Problog allow function symbols which means the grounding is infinite. MLNs have been extended to infinite domains [Singla and Domingos, 2007]. So-called nonparametric Bayesian approaches such as NP-BLOG [Carbonetto et al., 2005], infinite (hidden) relational models [Kemp et al., 2006, Xu et al., 2006], and relational Gaussian processes [Chu et al., 2006, Yu et al., 2006, Yu and Chu, 2007, Silva et al., 2007, Xu et al., 2009b] also allow for an unbounded number of individuals. These models use stochastic processes to specify how the random variables and their associated parameters are generated.

3.2 TWO EXAMPLE REPRESENTATIONS: MARKOV LOGIC AND PROBLOG

We now review two relational probabilistic model formalisms, which are probably most representative of the two main streams in StarAI: **Markov logic** and **ProbLog**. Markov logic [Richardson and Domingos, 2006] upgrades Markov network toward first-order logic, whereas ProbLog [De Raedt et al., 2007] is a probabilistic Prolog based on what has been called the distribution semantics [Poole, 1993b, Sato, 1995]. While Markov logic is a typical example of an undirected knowledge-based model construction, ProbLog is a directed model that can be seen as a relational probabilistic programming language. Furthermore, while Markov Logic is an example of an approach that extended a probabilistic graphical model with logic, ProbLog extends a logic programming language with probabilistic primitives. For implementations of Markov Logic and Problog as well as further materials on how to use them, see http://alchemy.cs.washington.edu/ and https://dtai.cs.kuleuven.be/problog/.

Both representations induce a probability distribution over possible worlds, where a world corresponds to a Herbrand model. The probability of a formula is the sum of the measures of the possible worlds in which it is true.

3.2.1 UNDIRECTED RELATIONAL MODEL: MARKOV LOGIC

Markov logic combines first-order logic with **Markov networks**. A Markov logic network consists of set of weighted first-order formulae. The probability of a world is proportional to the exponential of the sum of the formulae that are true in the world.

The idea is to view logical formulae as soft constraints on the set of possible worlds. A positive weight on a formula increases the probability of the worlds that satisfy the formula, and a negative weight on a formula decreases the probability of the worlds that satisfy the formula.

Example 3.8 Consider the following example (adopted from [Richardson and Domingos, 2006]). Friends & Smokers is a small Markov logic network that computes the probability of a person having lung cancer on the basis of her friends smoking. This can be encoded using the following weighted formulas:

$$\begin{array}{ll} 1.5: & cancer(P) \leftarrow smoking(P) \\ 1.1: & smoking(X) \leftarrow friends(X,Y) \wedge smoking(Y) \\ 1.1: & smoking(Y) \leftarrow friends(X,Y) \wedge smoking(X)\ . \end{array}$$

The first formula states the soft constraint that smoking implies cancer. So, interpretations in which people that smoke have cancer are more likely than those where they do not (under the assumptions that other properties remain constant). The second and third formulas state that friends of smokers are typically also smokers. The higher the weight of a formula, the stronger the constraint imposed by the formula is, and the more likely possible worlds satisfying the formula become.

A Markov logic network together with a set of constants $\{d_1, \ldots, d_k\}$ representing k individuals induces a grounded Markov network.

The random variables in the grounded network are the atoms in the Herbrand base of the form $p(d'_1, \cdots, d'_n)$ where p is a predicate and the d'_i are constants. For every ground instance $c_i\theta$ of a formula c_i in H, there is an edge between any pair of atoms $a\theta, b\theta$ that occur in $c_i\theta$.

The Markov network obtained by grounding the weighted formulae of Example 3.8 with the constants *anna* and *bob* is shown in Fig. 3.6.

The probability distribution over interpretations I is

$$\mathbf{P}(I) = \frac{1}{Z}\prod_{w:f} e^{w*n_f(I)} = \frac{1}{Z}e^{\sum_{w:f} w*n_f(I)}\ ,$$

$n_f(I)$ denotes the number of grounding substitutions θ for which $f\theta$ is satisfied by I, and Z is a normalization constant. Z is called the **partition function**.

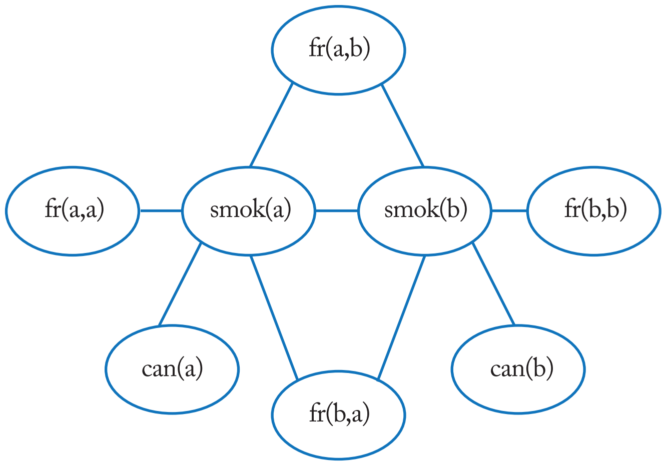


Figure 3.6: The Markov network induced by the Markov logic for the constants *a* and *b* (adapted from [Richardson and Domingos, 2006]).

There is also a product formulation of MLNs where the weights are all nonnegative and the probability of the world is the product of the formulae that are true in the world. The product formulation is more general as it allows zero-probability worlds. Zero-probability worlds can be handled in the log-formulation by having negative infinite weights (but positive infinite weights are problematic). The formulation in terms of the exponential function of a weighted sum is more common because it is easier to differentiate, which enables gradient-descent algorithms to be used for learning.

Note that for different (Herbrand) domains, different Markov networks will be produced. Therefore, one can view Markov logic networks as a template for generating Markov networks, and, hence, Markov logic is a knowledge-based model construction method.

It is often difficult to understand what the weights really mean, except in a few cases. For instance, if there is a single weighted formula, $w : f$, where all of the atoms in f share the same logical formula, the weight w is the log-odds of f, namely $\log P(f)/P(\neg f)$. When there are more complicated formulae, understanding the weights is more complicated because the weights are not local to the formula, but interact to give the appropriate probabilities.

Example 3.9 In Example 3.8, the formulae represent material implication. The weighted formulas of that example are equivalent to

$$\begin{aligned}
1.5 :& \quad cancer(P) \vee \neg smoking(P) \\
1.1 :& \quad smoking(X) \vee \neg friends(X, Y) \vee \neg smoking(Y) \\
1.1 :& \quad smoking(Y) \vee \neg smoking(X) \vee \neg friends(X, Y)\ .
\end{aligned}$$

The last formula adds the weight 1.1 to any model for every pair of individuals that are not friends or where one is smoking and one is not. Note that for any (ordered) pair of individuals, one of the last two formulae will be true. They are both true unless one of the individuals smokes, the other doesn't smoke and they are both friends. Note that when $X = Y$, both of the last formulae are true.

To illustrate how Markov logic generalizes logic we use an example due to [Richardson and Domingos, 2006]. Consider an MLN containing the single formula $r(X) \to s(X)$ with weight w, and a single constant a. This leads to four possible worlds/interpretations: $\{\neg r(a), \neg s(a)\}$, $\{\neg r(a), s(a)\}$, $\{r(a), \neg s(a)\}$, and $\{r(a), s(a)\}$. With this, we obtain that $P(\{r(a), \neg s(a)\}) = 1/(3e^w + 1)$ and the probability of each of the other three worlds is $e^w/(3e^w + 1)$. Thus, if $w > 0$, the effect of the MLN is to make the world that is inconsistent with $r(X) \to s(X)$ less likely than the other three. From the probabilities above we obtain that $P(s(a)|r(a)) = e^w/(1 + e^w)$. When $w \to \infty$, $P(s(a)|r(a)) \to 1$, recovering logical entailment. Furthermore, with changing domains, (e.g., by adding constants), the probabilities change as a result, and to keep the probabilities constant, the weights may need to change.

In summary, Markov logic and related formalisms have a number of important features.

- They are simple: an MLN is obtained by attaching weights to the formulae (or clauses) in a first-order knowledge base.
- They are intuitive in that the magnitude of the weight corresponds to the relative strength of its formula.
- In the infinite-weight limit, formulas reduce to first-order logic (over finite domains).
- They can be generalized to handle functions and in turn infinite domains [Singla and Domingos, 2007].
- They can naturally represent cyclic dependencies among variables.
- They can be viewed as log-linear models that use first-order formulae as feature templates. This allows them to compactly represent most graphical models (e.g., hidden Markov networks, probabilistic context-free grammars).
- In turn, many standard approaches for inference and learning log-linear models are easy to adapt to the MLN case.

3.2.2 DIRECTED RELATIONAL MODELS: PROBLOG

Here we describe a directed relational model in the spirit of probabilistic Horn abduction [Poole, 1991, 1993b], the independent choice logic (ICL) [Poole, 1997, 2008], Prism [Sato and Kameya, 1997, 2008], ProbLog [De Raedt et al., 2007], and CP-logic [Vennekens et al., 2009]. Here we will use the syntax of **ProbLog**.

Example 3.10 Consider a variant of the burglary alarm Bayesian network in Fig. 2.1 with multiple neighbors. The alarm depends on a burglary and whether there is an earthquake. Each neighbor can call independently given whether there is an alarm. To specify the conditional probability of alarm given burglary and earthquake, Problog uses rules and probabilistic facts:

$$\begin{array}{rl}
 & alarm \leftarrow burglary \wedge aifb. \\
 & alarm \leftarrow \neg burglary \wedge earthquake \wedge aifnbe. \\
 & alarm \leftarrow \neg burglary \wedge \neg earthquake \wedge aifnbne. \\
0.95 :: & aifb \\
0.29 :: & aifnbe \\
0.001 :: & aifnbne\ .
\end{array}$$

The first three rules have their normal meaning as they would in a logic program, with $\neg$ meaning negation as failure. Under negation as failure, an atom $\neg a$ is true when proving a fails in a possible world, cf. also Section 5.2.3. The first rule says that when *burglary* is true, *alarm* is true whenever *aifb* is true (*aifb* corresponds to the parameter of Bayesian network denoting the probability of *alarm* if *burglary* is true). The fourth line says that $P(aifb) = 0.95$. Together they specify $P(alarm \mid burglary) = 0.95$.

The probabilistic facts, such as the atom *aifb*, act as switches for the rules. When *aifb* is true, the alarm depends on the burglary if it is true. When *aifb* is false, it is the same as not having the rule.

For simple uses of probabilistic facts like above, these rules and probabilistic facts can be written more concisely as the following probabilistic rules:

$$\begin{array}{rl}
0.95 :: & alarm \leftarrow burglary \\
0.29 :: & alarm \leftarrow \neg burglary \wedge earthquake. \\
0.001 :: & alarm \leftarrow \neg burglary \wedge \neg earthquake\ .
\end{array}$$

This set of probabilistic rules is just an abbreviation for the earlier rules and probabilistic facts, but leaves the atoms that act as switches as implicit. For more complicated cases, such as where the switches are used in more than one rule, or when they are not parameterized by all of the logical variables, they can be made explicit.

This can be augmented to the relational case, where neighbors call as a function of whether there is an alarm:

$$\begin{array}{rl}
0.8 :: & calls(X) \leftarrow alarm \wedge neighbor(X). \\
0.1 :: & calls(X) \leftarrow \neg alarm \wedge neighbor(X). \\
 & neighbor(john). \\
 & neighbor(mary)\ .
\end{array}$$

This specifies that each neighbor calls as a function of whether there is an alarm. The sort of query we may want is

$$P(\mathit{burglary} \mid \mathit{calls}(\mathit{john}), \mathit{calls}(\mathit{mary})).$$

The syntax of Problog is defined in terms of:

- a **rule** is of the form $h \leftarrow b$, where h is an atom (the head of the rule) and b is a body, so a rule is a definite clause;
- a **probabilistic fact** is of the form $p :: a$, where p is a real number, $0 \leq p \leq 1$, and a is an atom, called an atomic hypothesis. No atomic hypothesis can unify with the head of a rule; and
- a **probabilistic rule** $p :: h \leftarrow b$ is an abbreviation for the rule $h \leftarrow b \wedge a$ and the probabilistic fact $p :: a$ for a new atom a that is parameterized by the logical variables in the rule.

When there is only a finite population, the semantics is defined by the following.

- There is a possible world for each selection of a subset of the ground instances of the atomic hypotheses. Let $\mathcal{A}(w)$ be the set of atomic hypotheses selected by world w.
- The atomic hypotheses are independent. Let $\mathcal{F}$ be the set of ground instances of the probabilistic facts. The probability of a world w in terms of the ground instances of the probabilistic facts is defined as follows:

$$P(w) = \prod_{\substack{p : a \in \mathcal{F} \\ a \in \mathcal{A}(w)}} p \prod_{\substack{p : a \in \mathcal{F} \\ a \notin \mathcal{A}(w)}} (1 - p) .$$

- The logic program together with the selected atoms $\mathcal{A}(w)$ specifies what is true in possible world w.
- The probability of a formula is defined in the standard way: the probability of a formula is the sum of the measures of the worlds in which the formula is true.

Example 3.11 Consider the following meaningless example:

$$\begin{aligned}
&0.1 :: a \\
&0.2 :: b \\
&0.3 :: c \\
&d \leftarrow a \wedge c, \\
&e \leftarrow b \wedge d, \\
&e \leftarrow \neg a, \\
&f \leftarrow \neg d \wedge \neg b .
\end{aligned}$$

The possible worlds are then:

$$\begin{array}{ccccccccl}
 & & \multicolumn{3}{c}{\overbrace{\qquad\qquad}^{\text{selection}}} & \multicolumn{3}{c}{\overbrace{\qquad\qquad\qquad}^{\text{logic program}}} & \\
w_0 & \models & a & b & c & d & e & \neg f & P(w_1) = 0.006 \\
w_1 & \models & a & b & & \neg d & \neg e & \neg f & P(w_1) = 0.014 \\
w_2 & \models & a & & c & d & \neg e & \neg f & P(w_1) = 0.024 \\
w_3 & \models & a & & & \neg d & \neg e & f & P(w_1) = 0.056 \\
w_4 & \models & & b & c & \neg d & e & \neg f & P(w_1) = 0.054 \\
w_5 & \models & & b & & \neg d & e & \neg f & P(w_1) = 0.126 \\
w_6 & \models & & & c & \neg d & e & f & P(w_1) = 0.216 \\
w_7 & \models & & & & \neg d & e & f & P(w_1) = 0.504\ .
\end{array}$$

$P(f) = 0.056 + 0.216 + 0.504 = 0.776$.

When there is arithmetic or function symbols, there can be infinitely many atomic hypotheses and the above semantics does not work; there are infinitely many worlds, each with probability 0. Just like with real-valued variables, we need a measure over *sets* of possible worlds (in what is called a sigma-algebra). The sets of possible worlds will be those that are described by logical formulae made up of atomic hypotheses. If S is a set of possible worlds described by a such a formula α, we define $\mu(S)$ as follows. The formula α can be put into the form $c_1 \vee \ldots \vee c_k$ where each c_i is a consistent conjunction of atomic hypotheses or their negation, and $c_i \cup c_j$ is inconsistent for $i \neq j$. Then $\mu(C) = \sum_i \prod_{a \in c_i} P(a) \prod_{\neg a \in c_i}(1 - P(a))$, where we treat a conjunction as a set of literals. We then define $P(h) = \mu(\{w \models h\})$, where here $\models$ means follows from the logic program. This semantics thus uses the set of atomic hypotheses that imply h. Finding the set of such hypotheses is called **abduction**. This explains why probabilistic Horn abduction [Poole, 1991, 1993b] was called that.

Example 3.12 To show how Problog works, even with cyclic programs,[1] consider the following representation of the friends and smokers example (where comments start with "%"):

$$\begin{array}{ll}
\quad\ \textit{smokes}(X) \leftarrow \textit{sm\_ind}(X). & \\
\quad\ \textit{smokes}(X) \leftarrow \textit{susceptible}(X) \wedge \textit{friends}(X,Y) \wedge \textit{smokes}(Y) \wedge \textit{sm\_fr}(X,Y) & \\
0.3 :: \textit{sm\_ind}(X). & \%X \text{ smokes independently of friends} \\
0.2 :: \textit{sm\_fr}(X,Y). & \%X \text{ smokes because of friendship with } Y \\
0.6 :: \textit{susceptible}(X). & \%X \text{ is susceptible to influence} \\
\quad\ \textit{friends}(X,Y) \leftarrow \textit{friends}(Y,X) \wedge \textit{fr\_symm}(X,Y) & \\
0.9 :: \textit{fr\_symm}(X,Y). & \%\text{symmetry makes } X \text{ and } Y \text{ friends} \\
\quad\ \textit{friends}(\textit{chris}, \textit{sam})\ . &
\end{array}$$

This says that a person X either smokes independently of their friends (when $\textit{sm\_ind}(X)$) or smokes when influenced by friends smoking. The atom $\textit{susceptible}(X)$ means that X is susceptible

[1] A program is cyclic if there can be atoms that depend on themselves. Problog does not allow negation in the cycle.

to be influenced by X's friends. For each smoking friend Y, the atom $sm\_fr(X, Y)$ means that X influences Y to smoke. The use of $fr\_symm(X, Y)$ allows us to model the fact that friendship is probably symmetric.

The fact that $sm\_fr(X, Y)$ is parameterized by Y means that each friend of X influences X independently, if X is susceptible. This implies that, in all the worlds where $sm\_ind(X)$ and $susceptible(X)$ are false, X does not smoke. Furthermore, the probability that X smokes is between 0.3 and $(0.3 + (1 - 0.3) * 0.6) = 0.72$; the first when there are no friends, and the latter is approached as the number of friends increases. The latter probability corresponds to the formula $\alpha = sm\_ind(X) \vee susceptible(X)$, which needs to be rewritten as $C = sm\_ind(X) \vee (\neg sm\_ind(X) \wedge susceptible(X))$ in order to make the conjunctions pairwise inconsistent.

Using probabilistic rules, the rules can be written as:

$$\begin{array}{ll}
0.3 :: & smokes(X). \\
0.2 :: & smokes(X) \leftarrow susceptible(X) \wedge friends(X, Y) \wedge smokes(Y). \\
0.6 :: & susceptible(X). \\
0.9 :: & friends(X, Y) \leftarrow friends(Y, X) \\
 & friends(chris, sam) \ .
\end{array}$$

This is an abbreviation for the above. [Note that when $smokes(X)$ is a probabilistic fact that unifies with the head of another rule, it must be interpreted as a probabilistic rule, not as a probabilistic fact.]

With 2 individuals, $chris$ and sam, there are 12 probabilistic atoms in the grounding (two instances each of $susceptible(X)$ and $sm\_ind(X)$, and four each of $sm\_fr(X, Y)$ and $fr\_symm(X, Y)$). Thus, there are 2^{12} possible worlds. Using the abductive characterization, we can partition the worlds into disjoint equivalence classes, so that each world in the same class has the same predictions for smokes, as in Fig. 3.7.

So $P(smokes(sam)) = 0.09 + 0.0252 + 0.1848 + 0.02268 = 0.32268$. One way to think about this is that $smokes(sam)$ is true iff

$$sm\_ind(sam) \vee (susceptible(sam) \wedge fr\_symm(sam, chris) \wedge sm\_fr(sam, chris) \wedge sm\_ind(chris)) \ .$$

That is, Sam smokes independently or is influenced by friend Chris.

If the propositions separated with an "or" are mutually exclusive in the formula α, their probabilities can be added. The above formula is equivalent to the conjunction C

$$\begin{aligned} sm\_ind(sam) \vee (&\neg sm\_ind(sam) \wedge susceptible(sam) \\ &\wedge fr\_symm(sam, chris) \wedge sm\_fr(sam, chris) \wedge sm\_ind(chris)) \end{aligned}$$

which has disjunction of exclusive propositions, and so the probabilities can be added. So $P(smokes(sam)) = \mu(C) = 0.3 + (1 - 0.3) * 0.6 * 0.9 * 0.2 * 0.3 = 0.32268$.

As already illustrated, probabilistic facts (or binary switches) are expressive enough to represent a wide range of models, including Bayesian networks, Markov chains, and hidden Markov

Worlds that select	Smoking prediction		Probability
	Chris	Sam	
$sm\_ind(chris) \wedge sm\_ind(sam)$	true	true	0.09
$\neg sm\_ind(chris) \wedge \neg sm\_ind(sam)$	false	false	0.49
$\neg sm\_ind(chris) \wedge sm\_ind(sam) \wedge susceptible(chris) \wedge sm\_fr(chris, sam)$	true	true	0.0252
$\neg sm\_ind(chris) \wedge sm\_ind(sam) \wedge (\neg susceptible(chris) \vee \neg sm\_fr(chris, sam))$	false	true	0.1848
$sm\_ind(chris) \wedge \neg sm\_ind(sam) \wedge susceptible(sam) \wedge fr\_symm(sam, chris) \wedge sm\_fr(sam, chris)$	true	true	0.02268
$sm\_ind(chris) \wedge \neg sm\_ind(sam) \wedge (\neg susceptible(sam) \vee \neg fr\_symm(sam, chris) \vee \neg sm\_fr(sam, chris))$	true	false	0.18732

Figure 3.7: Partitioning of possible worlds for Example 3.12.

models. However, for ease of modeling, it is often more convenient to use multi-valued random variables instead of binary ones. This is incorporated in languages such as the independent choice logic (ICL) [Poole, 1997, 2008], Prism [Sato and Kameya, 1997, 2008], ProbLog [De Raedt et al., 2007], and CP-logic [Vennekens et al., 2009] in different ways. Here we will use the annotated disjunctions concept of Vennekens et al. [2004], which essentially extends the concept of a probabilistic rule, and the exposition in [De Raedt and Kimmig, 2015].

An **annotated disjunction** (AD) is an expression of the form

$$p_1 :: h_1; \; \ldots \; ; p_N :: h_N \leftarrow b_1, \; \ldots \; , b_M . \, ,$$

where $b_1, \ldots, b_M$ is a possibly empty conjunction of literals, the p_i are probabilities and $\sum_{i=1}^{N} p_i \leq 1$. Considered in isolation, an annotated disjunction states that if the body $\mathtt{b}_1, \ldots, \mathtt{b}_M$ is true, h_i is true with probability p_i. This rules makes at most one of the h_i true. If the p_i in an annotated disjunction do not sum to 1, there is also the case that nothing is chosen. The probability of this event is $1 - \sum_{i=1}^{n} p_i$.

A probabilistic fact is thus a special case of an AD with a single head atom and empty body. A probabilistic rule is then a special case of an AD with a single head atom.

For instance, consider the following AD:

$$\frac{1}{3} :: color(B, green); \frac{1}{3} :: color(B, red); \frac{1}{3} :: color(B, blue) \leftarrow ball(B) \; .$$

It defines an independent probabilistic choice of color for each ball B, and basically states that the color of each ball B is uniformly distributed over the values *green*, *red*, and *blue*. As for probabilistic facts, a non-ground AD denotes the set of all its groundings, and for each such grounding, choosing one of its head atoms to be true is seen as an independent random event.

The probabilistic choice over head atoms in an annotated disjunction can equivalently be expressed using a set of logical clauses, one for each head, and a probabilistic choice over facts added to the bodies of these clauses, e.g.,

$$\begin{array}{l}
color(B, green) \leftarrow ball(B), choice(B, green). \\
color(B, red) \leftarrow ball(B), choice(B, red). \\
color(B, blue) \leftarrow ball(B), choice(B, blue). \\
\frac{1}{3} :: choice(B, green); \frac{1}{3} :: choice(B, red); \frac{1}{3} :: choice(B, blue)\ .
\end{array}$$

This example illustrates that annotated disjunctions define a distribution over basic facts as before, but can simplify modeling by directly expressing probabilistic consequences. It is only that the probabilistic choices are now mutually exclusive. Furthermore, it can be shown that ADs can be mapped onto an equivalent Problog program without ADs.

In summary, Problog and related formalisms have a number of important features.

- They can represent any Bayesian network with a local translation, where the number of atomic hypotheses is equal to the number of parameters (or CPT entries) that need to be specified.

- The rules can be interpreted as in ordinary logic programs, with the normal meaning. Rules with probabilities can be interpreted as ordinary logical rules with an additional atom in the body.

- They are naturally relational in the sense that Prolog or Datalog is relational.

- They can represent context-specific independence, as in Example 3.10. The "or" combination of logic programs together with the probabilistic facts gives the standard "noisy-or" combination rule [Pearl, 1988].

- They can represent cyclic networks with an intuitive semantics; see Example 3.12. Note that while Problog can also use negation as failure in the rules, negation as failure and cyclicality cannot be mixed arbitrarily. We cannot have $a \leftarrow \neg b$ and $b \leftarrow a$; see Section 5.2.3 for more details.

- Using annotated disjunction extensions, it is convenient to reason about multi-valued random variables and switches.

CHAPTER 4

Representational Issues

When dealing with complex domains, there are many diverse pieces of information that should be taken into account for an informed decision, and there are diverse needs for data. This explains why there are so many different probabilistic relational modeling approaches, of which we reviewed MLNs and ProbLog. Rather than listing and describing more of the various proposals, we here describe some issues that arise in designing a representation language.

4.1 KNOWLEDGE REPRESENTATION FORMALISMS

Knowledge representation [Sowa, 2000, Brachman and Levesque, 2004] is the study of how knowledge can be expressed, reasoned with, and learned. There are a number of different expectations of the role of knowledge representation that are often at odds when combining logic and probability.

- *KR as semantics*: we want to devise formalisms in which one can state whatever one wants, specify what a theory in this formalism means or entails, and then derive its logical consequences. For example, the logics of Bacchus [1990] and Halpern [1990] combined logic and probabilistic reasoning. The problem with such logics is that we have to state much information and the answers are often very weak, either saying "I do not know" or giving only weak probability bounds.

- *KR as common-sense reasoning*: we want a system where one can throw in any knowledge and get out "reasonable" answers. This was the motivation behind nonmonotonic reasoning [McCarthy and Hayes, 1969], maximum entropy approaches [Jaynes, 2003], and undirected models such as Markov networks [Pearl, 1988]. The problem with such approaches is that the knowledge is not modular; it is often difficult to predict the consequences of adding a single piece of knowledge.

- *KR as modeling*: we want a symbolic modeling language for "natural" generative modeling of domains. Some examples include logic programming [Kowalski, 2014] and Bayesian networks [Pearl, 1988]. In both of these representations, for everything modeled we need to specify what it depends on (either in terms of rules or conditional probabilities). The problem with such representations is that we cannot just add arbitrary pieces of knowledge, but need to systematically state our knowledge. Missing information has a particular meaning, which is not just that the knowledge is unknown.

Many of the different proposals can be seen as fitting into one of these frameworks.

Example 4.1 To see how these expectations clash, consider a probabilistic model with two logical variables A and B, where a means $A = true$, and b means $B = true$. Suppose we are given $P(a) = 0.5$ and $P(b) = 0.5$, and and that is all the information we have. We would like to know $P(a \mid a \vee b)$. In the KR as commonsense reasoning view, one might assume a maximum entropy solution, and so answer $P(a) = 2/3$. (The maximum entropy solution does not even require us to specify $P(a) = 0.5$ and $P(b) = 0.5$.) In the *KR* as semantics, the answer might be $P(a)$ is in the range $[0.5, 1.0]$ on the grounds that if A and B are perfectly correlated, the answer is 1, and if they are perfectly negatively correlated, the answer is 0.5. As the dependence has not been specified, we need to give the range. In the KR as modeling formulation view, just specifying $P(a)$ and $P(b)$ is not a legal model; we also need to specify the interdependence between A and B. All of the above cases, with A and B explicitly independent, or with any value in the range $[0.5, 1.0]$ by varying how A and B are dependent, can be modeled. The model makes the dependence explicit.

4.2 OBJECTIVES FOR REPRESENTATION LANGUAGE

Often a good representation is a compromise between many competing objectives. Some of the **objectives of a representation language** include the following.

Expressivity: The representation language should be rich enough to represent the knowledge required to solve the problems at hand. A generally useful representation (as opposed to a special-purpose representation) should deal with discrete and continuous random variables, with domains evolving over time. Ultimately, it should be able to represent an unbounded or infinite set of individuals.

Efficient inference: It should be able to represent structure that enables a reasoner to solve problems efficiently, in the worst case or on average for some distribution of problems.

Understandability or explainability: It should be easy to understand models in the language. For a system to be relied on for critical life-or-death decisions or ones that involve large economic risks, the system needs to be able to explain or at least justify its recommendations to the people who are legally responsible.

Learnability: It should facilitate the learning of models from examples. The learned models should generalize well beyond the training set. This is desired for all of machine learning, but it is compounded in relational models because we may have one big data set, for example, all of the student records in a university, and we want to make predictions about those students in existing courses, as well as new students in existing courses, existing students in new courses or about other students in other courses.

Compactness: The representation of knowledge should be succinct. This is not only related to the ease of modeling but also related to prediction accuracy, as it is conjectured that compact representations generalize better, and to explainability as a compact models can easier be explained.

Modularity: Parts of the representation/model should be understood and/or learned separately. This has two aspects.

- The ability to conjoin models that are developed independently. If different parts of a model are developed separately, can these parts easily be combined? This should be possible even when the people developing the component models do not know their work will be combined and even if some parts are directed and some parts are undirected.
- The ability to decompose a model into smaller parts for specific applications. If there is a big model, can someone take a subset of the model and use it? Can someone take advantage of a hierarchical decomposition leading to weakly interacting parts [Simon, 1996]?

Ability to incorporate prior knowledge: If there is domain knowledge, it should be easy to be incorporated into the model. Here, the prior should be refutable, in the sense that if there is enough evidence against it, the evidence will eventually dominate. At the other extreme, if there is very little data, the prior should dominate, and there should be a smooth transition between these.

In the big data case, as the amount of data grows, so does the complexity of each individual. For example, even if we were to have the electronic health records of every person on earth, there will be very few pairs of people with identical descriptions, and only then when there is very little information about the individuals. Even in this case, we cannot assume that the data overwhelms the prior.

Interoperability with heterogenous data: The representation should be learnable from multiple heterogeneous data sets. Heterogeneous can mean having different overlapping concepts, derived from different contexts, at different levels of abstraction (in terms of generality and specificity) or different levels of detail (in terms of parts and subparts). The levels of abstraction and detail can even vary among individuals in the same data set. It should also be able to incorporate causal data that has arisen from interventions [Pearl, 2009] and observational data that only involves passively observing the world. For example, in medicine there is data from randomized clinical trials and large amounts of observational data from health records that need to be combined. Some patients have huge amounts of information available about them, such as the results of multiple observations and tests at multiple times, and some have virtually none.

Latent variables: The representation should be able to learn the relationship between parametrized random variables if the relationship is a function of observed and unobserved (latent) variables. Latent variables typically make models simpler and facilitate compactness but may sacrifice understandability. In relational models, latent variables can include the hypothesis of new or previously unobserved individuals.

Of course, not all of the existing representations have attempted to achieve all of these desiderata. In the following we will examine some issues that have arisen in the various representations and how they have been addressed. Whether these issues turn out to be unimportant or whether one choice will ultimately prevail is an open question.

4.3 DIRECTED VS. UNDIRECTED MODELS

In a manner similar to the difference between Bayesian networks and Markov networks [Pearl, 1988], there are **directed relational models** (e.g., [Poole, 1997, 2008, Heckerman et al., 2004, Laskey, 2008]) and **undirected relational models** (e.g., [Richardson and Domingos, 2006]). The main differences between these are as follows.

- The probabilities of the directed models can be directly interpreted as conditional probabilities. This means that they can be explained to users, and can be locally learned if all of the relevant properties are observed. The weights of the undirected models cannot be locally interpreted; all of the weights act together to produce a probability.
- Because the probabilities can be interpreted locally, the directed representations are modular. The probabilities can be acquired modularly, and need not change if the model is expanded or combined with another model.
- For inference in directed models, typically only a very small part of the grounded network needs to be used for inference. In particular, only the ancestors of the observed and query relations need to be considered. This has been characterized as abduction [Poole, 1993a], a logical inference method that involves finding consistent sets of assumable atoms that logically imply some observations. On models where there are probabilities over atoms (see Section 4.6), abduction can be used to extract the relevant probabilities from proofs of the observations and queries.
- The directed models typically require the structure of the conditional probabilities to be acyclic. For relational models, this is problematic if we have conditional probabilities of the form:

  $$P(foo(X) \mid foo(Y) \ldots) .$$

  To make this acyclic, we could totally order the individuals, and enforce a "less than" operation. However, this means that the individuals are no longer exchangeable, because swapping the individuals will affect the ordering. One example where this arises is if we wanted to specify the probability that friendship is transitive:

$$P(friends(X, Y) \mid \#Z \text{ s.t. } friends(X, Z), friends(Z, Y)) ,$$

which specifies the probability that X and Y are friends as a function of the number of friends they have in common.

Naive ways to incorporate such rules into directed models often do not work as expected because the grounding of the statements are typically not consistent, but are sometimes interpreted in terms of the equilibrium of a Markov chain [Neville and Jensen, 2007]. But ProbLog allows for cyclic statements as it is based on a logic programming semantics, see Section 5.2.3 for a detailed explanation. Using the least Herbrand Model semantics, the cycle $a \leftarrow b$ and $b \leftarrow a$, with the intuitive interpretation that a and b have the same truth value, will have a preference for both a and b being false (as in Example 3.12). But, as explained in Section 5.2.3, dealing with negation as failure for such cycles is more complicated.

- Undirected models, such as Markov logic networks [Richardson and Domingos, 2006], do not have a problem with acyclicity. In applications where a relation for some individual depends on the same relations to other individuals (such as in the transitivity example above), the undirected models have been empirically more effective.

- For simple undirected models, the weights of the undirected models can often be interpreted as log-odds. However, for more complex models, weights of the worlds involve complex interactions between variables, there are many ways to represent the same distribution, and the log-odds interpretation does not work. It is then often very difficult to interpret the numbers.

- Ways for directed models to handle cyclic dependencies and ways for undirected models to be modular and explainable are open research problems.

For the non-relational cases, directed models and undirected models can be translated into each other. Unfortunately, this is not true for the relational case.

- Directed non-relational models can be translated into undirected models in a modular way. To represent $P(A \mid BCD)$, we can "marry" the parents and treat this as a potential on $ABCD$. For some cases, the structure of the rules goes through. For example, the Problog-style rule:

$$p :: a \leftarrow b \wedge c$$

becomes the two weighted formulas:

$$\begin{aligned} p &: a \wedge b \wedge c. \\ 1 - p &: \neg a \wedge b \wedge c \end{aligned}$$

as long as the rules for a have mutually exclusive bodies. Surprisingly, directed relational models cannot be translated into undirected models such as MLNs. Undirected relational models are provably **non-modular**; Buchman and Poole [2015] proved that it is impossible to write an **aggregator** in MLNs (without quantification) that represents $P(a \mid b(X))$ without introducing side effects in the b's. Any way you might try to represent this conditional probability in MLNs, as long as the b's affect a, they affect each other when a is not observed. Quantifiers in the weighted formula allow for a limited number of aggregators that do not induce side effects (e.g., we can represent "or" but not noisy-or without latent variables).

- Undirected models can also be translated into directed models. Given a weighted formula $w_i : f_i$, we can create a rule $t_i \leftarrow f_i$ with weight 2^{w_i} (suitably scaled so all of the weights are less than one), and then condition on each of the $t_i = true$. This also works for relational models.

4.4 FIRST-ORDER LOGIC VS. LOGIC PROGRAMS

There is a fundamental difference between working with first-order clausal logic and with logic programs. The difference is that a set of first-order formulas may have multiple models, whereas the semantics of a logic program is given by its least Herbrand model. This difference is important as it determines not only what can be expressed but also has important implications for probabilistic representations designed on top of these logics. To illustrate these differences we use an example due to [De Raedt and Kimmig, 2015]:

$$\begin{gathered} edge(1,2). \\ path(A,B) \leftarrow edge(A,B). \\ path(A,B) \leftarrow edge(A,C), path(C,B)\ . \end{gathered}$$

The least Herbrand model of the *path* program is $\{edge(1,2), path(1,2)\}$, which corresponds to the transitive closure of the predicate *edge*. The transitive closure of a relation which contains the relation itself, is transitive and is minimal.

It is well known in knowledge representation that the transitive closure cannot be expressed in first-order logic, that one either needs second-order logic, or a least Herbrand model semantics. Indeed, when interpreting the above clauses in first-order logic and not as a logic program, the following interpretations are also models of the theory:

$$\begin{gathered}
\{edge(1,2), path(1,2)\} \\
\{edge(1,2), path(1,2), path(1,1)\} \\
\{edge(1,2), path(1,2), path(2,1)\} \\
\{edge(1,2), path(1,2), path(2,2)\} \\
\ldots
\end{gathered}$$

It can be seen that while path is transitive in each of these models and possible worlds, the transitive closure property only holds in the first model.

Even though this example is purely logical, it also illustrates a key difference between Markov Logic and Problog. Viewed as a Problog program, there would be only one possible world given by the least Herbrand model. This implies that facts in the least Herbrand model have a probability of 1, and all other ground facts have probability 0. Viewed, however, as a Markov Logic Network, each clause would need to get an infinite weight as the clauses are required to hold in all possible worlds and the clauses act as hard constraints. Furthermore, as there are no soft constraints, Markov Logic will assign equal probabilities to all of these possible worlds, which is in line with the maximum entropy principle. As a consequence, while the probabilities of $edge(1,2)$ and $path(1,2)$ are still equal to 1, the probabilities of $path(1,1)$ and $path(2,2)$ is no longer 0 as in Problog, but is equal to the proportion of possible worlds in which they are true.

4.5 FACTORS AND FORMULAE

In graphical models, a **factor** or **potential** [Shafer and Shenoy, 1990, Zhang and Poole, 1994, Frey et al., 1997] on a set of random variables represents a function from the ranges of the variables into other values (typically reals). For example, a factor on random variables $\{A, B, C\}$ is a function from $range(A) \times range(B) \times range(C) \to \Re$, where $range(X)$ is the range of random variable X, and $\Re$ is the set of real numbers. Factors can represent conditional probabilities, utilities, potentials for undirected models, as well as the intermediate results of computation.

An alternative to using an explicit representation of full factors is to represent the value of one assignment to variables, or to a Boolean function of assignments to values, such as using clauses [Darwiche, 2002]. This enables local structure to be exploited including **context-specific independence** and zero parameters [Chavira and Darwiche, 2005, Sang et al., 2005]. For example, we may represent a factor on random variables $\{A, B, C\}$ as weighted formulae, of the form *formula* : *weight* such as:

$$\begin{aligned}
&a : w_0 \\
&\neg a \wedge b : w_1 \\
&\neg a \wedge \neg b \wedge c : w_2 \\
&\neg a \wedge \neg b \wedge \neg c : w_3 \ ,
\end{aligned}$$

where w_0 is used whenever A is true, and w_1 is used when A is false and B is true, etc.

In relational representations the representation corresponding to a factor is a parameterized factor or **parfactor** [Poole, 2003, Milch et al., 2008], which is a triple

$$\langle C, V, \phi \rangle ,$$

where C is a set of inequality constraints on logical variables, V is a set of parameterized random variables and ϕ is a factor on V. Factor ϕ is used for all assignments of individuals to logical variables in V.

Example 4.2 Consider the grades example shown in Fig. 3.3. This could be represented as parfactors:

$$\begin{array}{l}\langle \{\}, \{int(S)\}, \theta_i \rangle \\ \langle \{\}, \{diff(C)\}, \theta_d \rangle \\ \langle \{\}, \{int(S), diff(C), grade(S, C)\}, \theta_g \rangle ,\end{array}$$

where θ_i specifies a real number for each value in the range of *int* which sum to one (so if *int* is Boolean, θ_i specifies two non-negative real numbers). Similarly, θ_g gives a real number for each assignment of a value to $int(S), diff(C), grade(S, C)$, which can represent the conditional probability.

Example 4.3 Consider representing the relation $has\_motive(X, Y)$ in Fig. 3.4, which is true when person X has a motive to shoot person Y. We would expect that the probability that someone has a motive to shoot themselves ϕ_1 would be different to the probability that they have a motive to shoot someone else ϕ_2. Thus, we may expect the following parfactors to represent the prior probability:

$$\begin{array}{l}\langle \{\}, \{has\_motive(X, X)\}, \phi_1 \rangle \\ \langle \{X \neq Y\}, \{has\_motive(X, Y)\}, \phi_2 \rangle .\end{array}$$

As with the propositional case, an alternative representation is to use formulae with free variables. Markov Logic Networks [Richardson and Domingos, 2006] use weighted logical formulae with free logical variables. Problog and related systems [Poole, 1993b, Sato and Kameya, 1997, De Raedt et al., 2007] allow for multiple rules to define conditional probabilities. These representations allow the specification of context-specific independencies by either including atoms in the rules or not. The directed models represent **zero probabilities** by omitting formulae, and the undirected models can represent zero probabilities using negative infinity weights. Context-specific independencies and zero probabilities can be exploited computationally.

4.6 PARAMETERIZING ATOMS

Poole [1991, 1993b] noticed that a language does not need to allow probabilities over arbitrary formulae; probabilities over atomic formulae (atoms) was adequate to represent any (conditional) probability distribution, as long as there is a logical knowledge base which specifies the consequences of the atoms being true. Moreover, these atoms can be assumed to be unconditionally independent. In this **independent choice** model, the semantics can either be defined in terms of independent choices and a logic program that gives the consequences of choices, or by an "abductive" semantics where only the atoms required to prove a proposition are required.

More recently, inference methods, based on MLNs and **model counting** [Gogate and Domingos, 2011, Van den Broeck et al., 2011], have used probabilities on atoms to allow probabilistic inference to be implemented by model counting. These methods, although superficially similar, have very different properties; consider the following example.

Example 4.4 Suppose we want to have rules with probabilities:

$$\begin{aligned} &a \leftarrow b : p_1 \\ &a \leftarrow b \wedge c : p_2 \ . \end{aligned}$$

Both methods create atoms n_1 and n_2 with $P(n_1) = p_1$ and $P(n_2) = p_2$. The independent choice model results in the rules:

$$\begin{aligned} &a \leftarrow b \wedge n_1 \\ &a \leftarrow b \wedge c \wedge n_2, \end{aligned}$$

which means its completion [Clark, 1978], just as in:

$$a \leftrightarrow (b \wedge n_1) \vee (b \wedge c \wedge n_2) \ .$$

The completion of the predicate a is based on the observation that there are two clauses for the predicate a, and so a will be true if the disjunction of the condition parts of these clauses is true. The completion is then obtained by replacing the implication ($\leftarrow$) by an equivalence ($\leftrightarrow$).

The method of Gogate and Domingos [2011], Van den Broeck et al. [2011] (which we will call the MLN method) would result in:

$$\begin{aligned} &(a \leftarrow b) \leftrightarrow n_1 \\ &(a \leftarrow b \wedge c) \leftrightarrow n_2 \ . \end{aligned}$$

In the independent choice method, n_1 and n_2 can be independent. In the MLN method, n_1 and n_2 cannot be independent as n_1 logically implies n_2. In the independent choice method p_1 and p_2 can be arbitrarily assigned, whereas in the MLN method, $P(n_1) \geq P(n_2)$.

What seems to be a slight difference in the reading of rules has a big effect on the semantics. The independent choice method allows a simple semantics in terms of independent atomic choices and a logic program that gives the consequences of the choices [Poole, 1993b, 1997]. The MLN method [Gogate and Domingos, 2011, Van den Broeck et al., 2011] cannot assume independent atoms, but appeals to the maximum entropy assignment, which makes it much more difficult to interpret the numbers.

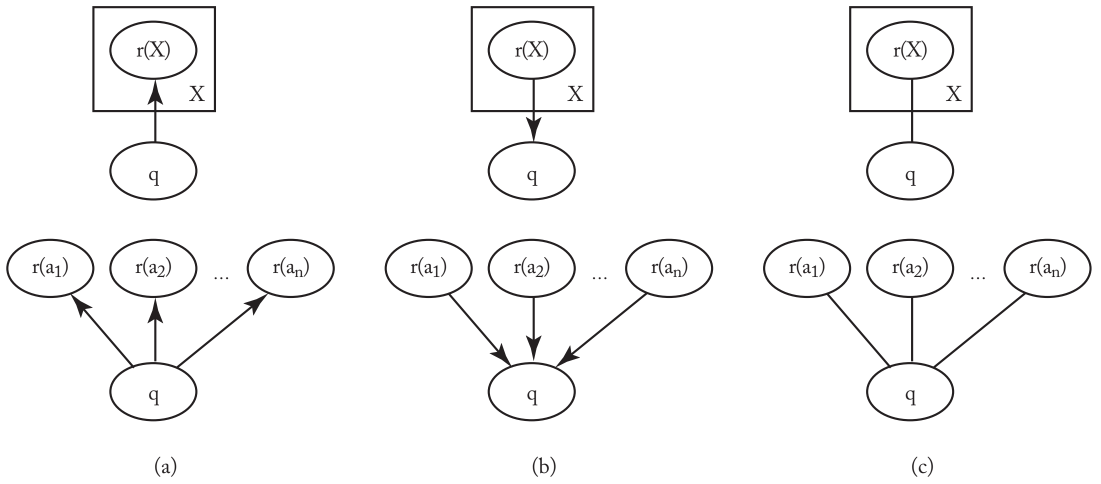


Figure 4.1: Running example as (a) naïve Bayes, (b) logistic regression (with independent priors for each $r(X)$), and (c) Markov network. On the top are the parametrized networks, and on the bottom are the the groundings for the population $\{a_1, a_2, \ldots, a_n\}$.

4.7 AGGREGATORS AND COMBINING RULES

Suppose binary random variable q is connected to binary parametrized random variable $r(X)$ which contains an extra logical variable, X, as in Fig. 4.1. In the grounding, q is connected to an unbounded number of instances of $r(X)$. A directed model where $r(X)$ is a child of q, as in Fig. 4.1a, produces a naive Bayesian model in the grounding. An undirected model with a potential for q and a pairwise potential for each factor, as in Fig. 4.1c, gives the same model as Fig. 4.1a, but without the restriction that summing out an instance of r has no effect. In both of these models the joint probability is the product of factors. However, for the directed model with $r(X)$ as a parent of q (Fig. 4.1b), in the grounding q has an unbounded number of parents. As we require a finite representation, there must be some way to aggregate the parents.

There are two main approaches to deal with this "multiple-parent" problem: **aggregators** and **combining rules**. Aggregators define how the probability of the child depends on the values for the parents, and combining rules define how the probability of the child is a combination of the probabilities of the parents. Common ways to aggregate in relational models [Horsch and Poole, 1990, Friedman et al., 1999, Neville et al., 2005, Perlich and Provost, 2006] include logical operators such a noisy-or, logistic regression, or ways to combine the probabilities. This requirement for aggregation occurs in a directed model whenever a parent contains an extra logical variable.

For a positive model (with no zero probabilities) the aggregation model that corresponds to the naive Bayesian model or an MLN is a **logistic regression** model.

In particular, suppose that the joint probability is a product of non-negative factors:

$$P(Q, R_1, \ldots, R_n) \propto \prod_i f(Q, R_i) \times g(Q) ,$$

where Q and R_i are binary. In terms of the models of Fig. 4.1, R_i is $r(a_i)$ for some enumeration $a_1, \ldots, a_n$ of the population of size n. This form covers Markov networks, naïve Bayes, and any model where the probability is proportional to the product of (pairwise) factors.

Choose a particular value q for Q, and observations for values of the R_i, writing $\neg q$ as the negation of assignment q:

$$\begin{aligned} P(q|R_1, \ldots, R_n) &= \frac{P(q, R_1, \ldots, R_n)}{P(q, R_1, \ldots, R_n) + P(\neg q, R_1, \ldots, R_n)} \\ &= \frac{1}{1 + \frac{P(\neg q, R_1, \ldots, R_n)}{P(q, R_1, \ldots, R_n)}} \\ &= \frac{1}{1 + 1/\left(\prod_i f(q, R_i)/f(\neg q, R_i) \times g(q)/g(\neg q)\right)} \\ &= \frac{1}{1 + e^{-\sum_i \hat{f}(q, R_i) + \hat{g}(q)}} , \end{aligned}$$

where $\hat{f}(q, R_i) = \log(f(q, R_i)/f(\neg q, R_i))$ and $\hat{g}(q) = \log(g(q)/g(\neg q))$. This last step is only valid if all potentials are positive (none are zero).

With a fixed way to represent *false* and *true* for R_i (e.g., as $\{0, 1\}$ or $\{-1, 1\}$), we can then represent $\hat{f}(q, R_i)$ as $w_i R_i$ and $g(\hat{q})$ as w_0 (since q is fixed). Thus, we have

$$P(q \mid R_1, \ldots, R_n) = \text{sigmoid}\left(w_0 + \sum_i w_i R_i\right) , \tag{4.1}$$

where sigmoid$(\cdot)$ is the sigmoid or logistic function:

$$\text{sigmoid}(x) = 1/(1 + e^{-x}) .$$

The space of assignments to the w_i so that $w_0 + \sum_i w_i R_i = 0$ is called the **decision threshold**. It is the boundary of where $P(q \mid R_1, \ldots, R_n)$ changes between being closer to 0 and being closer to 1. Equation (4.1) implies $P(q \mid R_1, \ldots, R_n) > 0.5$ iff $w_0 + \sum_i w_i R_i > 0$.

For a relational model where the individuals are exchangeable, w_i must be identical for all variables R_i, so (4.1) becomes:

$$P(q \mid R_1, \ldots, R_n) = \text{sigmoid}\left(w_0 + w_1 \sum_i R_i\right) . \tag{4.2}$$

The following example considers what happens with a relational model, in which n can vary [Poole et al., 2012].

Example 4.5 Suppose we want to represent "q is true if and only if r is true for 5 or more individuals" (i.e., $q \equiv |\{i : R_i = true\}| \geq 5$) using a logistic regression model ($P(q) \geq 0.5) \equiv$

Table 4.1: Prediction for a population of 20 as a function of how *True* and *False* are represented. These models make the same predictions for a population of 10. See Example 4.5.

false	true	w_0	w_1	Prediction for $n = 20$
0	1	-4.5	1	$Q \equiv \lvert\{R_i = true\}\rvert \geq 5$
-1	1	0.5	0.5	$Q \equiv \lvert\{R_i = true\}\rvert \geq 10$
-1	0	5.5	1	$Q \equiv \lvert\{R_i = true\}\rvert \geq 15$
-1	2	$-\frac{7}{6}$	$\frac{1}{3}$	$Q \equiv \lvert\{R_i = true\}\rvert \geq 8$
1	2	-14.5	1	$Q \equiv \lvert\{R_i = true\}\rvert \geq 0$

$(w_0 + w_1 \sum_i R_i \geq 0)$, which we fit for a population of 10. Consider what this model represents when the population size is 20, as a function of how *True* and *False* are represented.

- If *False* is represented by 0 and *True* by 1, this model will have q true if r is true for 5 or more individuals out of a population of 20. It is easy to see this, as $\sum_i r_i$ only depends on the number of individuals for which r is true.
- If *False* is represented by -1 and *True* by 1, this model will have q true if r is true for 10 or more individuals out of a population of 20. The sum $\sum_i r_i$ depends on how many more individuals have r true than have r false.
- If *True* is represented by 0 and *False* by any other value, this model will have q true if r is true for 15 or more individuals out of a population of 20. The sum $\sum_i r_i$ depends on how many individuals have r false.

Other parametrizations can result in different decision thresholds. Table 4.1 gives some possible parameter settings as a function of the numerical representation of *false* and *true*, which represent the same conditional distribution for $n = 10$, and the corresponding prediction for a population size of 20.

The way the decision threshold grows with the population size n does not depend on data (which would provide the weights), but on the prior assumptions, which are implicitly encoded into the numerical representation of R_i.

While the dependence on population may be arbitrary when a single population is observed, it affects the ability of a model to predict when multiple populations or subpopulations are observed.

Example 4.6 Suppose we want to model whether someone is happy and the happiness depends on the number of their friends that are mean to them. One model could be that people are happy as long as they have more than five friends who are not mean to them. Another model could be that people are happy if more than half of their friends are not mean to them. A third model could be that people are happy as long as fewer than five of their friends are mean to them. These three models coincide for people with 10 friends, but make different predictions for people with 20 friends.

A particular logistic regression (or a naive Bayesian) model is only able to model one of the dependencies of how predictions depend on population, and so cannot properly fit data that does not adhere to that dependence. The dependence on the representation of *true* and *false* can be avoided by always including the property 1 (which has the value 1 for each individual; in the naive Bayesian model, this needs to be observed) or have separate parameters for the positive and negative cases. These two methods are equivalent representations, with weights that can be mapped, e.g.,

$$\begin{aligned} P(q \mid R_1, \ldots, R_n) &= \operatorname{sigmoid}\left(w_0 + w_1 \sum_i R_i + w_2 \sum_i (1 - R_i)\right) \\ &= \operatorname{sigmoid}\left(w_0 + (w_1 - w_2) \sum_i R_i + w_2 n\right) . \end{aligned}$$

These models are flexible enough to fit all of the cases in Example 4.6. Markov logic networks allow for arbitrary polynomial growth in the sigmoid, e.g., the formula $q \wedge r(X) \wedge r(Y)$ allows for squared growth with the number of positive instances. However, the three models of Fig. 4.1 have very different characterizations as the population changes, even with no observations [Poole et al., 2012]. In the naive Bayes model, neither Q nor R depend on the population size. In the logistic regression model, $P(r(a_i))$ does not depend on the population, but $P(q)$ does. In the Markov network, both $P(r(a_i))$ and $P(Q)$ depend on the population.

Another way that has been suggested to specify aggregators is in terms of rules that combine individual probabilities [Jaeger, 2007, Natarajan et al., 2009, Kersting and De Raedt, 2007]. Each parent or set of related parents produces a value for the child, all of which are combined using a deterministic or stochastic function.

Example 4.7 Suppose $q(X, Y)$ and $s(X, Y)$ are the parents of $r(X)$, where the the population of Y is $\{y_1, \ldots, y_n\}$. For each i, we have a distribution $P(r(x_1) \mid q(x_1, y_i), s(x_1, y_i))$, denoted as p_i. All of these p_i's can be combined using a deterministic or stochastic function f to obtain the distribution,

$$P(r(x_1) \mid q(x_1, y_1), s(x_1, y_1), \ldots, q(x_1, y_n), s(x_1, y_n)) = f(p_1, \ldots p_n) ,$$

where f is called the combining rule.

Not all ways of combining probabilities give coherent probabilistic models. Natarajan et al. [2010] present an algorithm for converting a directed model with combining rules to an equivalent MLN. They showned that for a particular class of combining rules called *decomposable combining rules*, the resulting distribution is coherent (there is an equivalent class of aggregators). The most common combining rules used in the literature are **mean**, **weighted mean**, and **noisy-or** [Natarajan et al., 2009, Jaeger, 2007, Kersting and De Raedt, 2007]. For these combining rules there are equivalent aggregators. For example, mean can be represented as a stochastic choice using a uniform distribution over the parent values, weighted mean corresponds to a stochastic choice given the weight distribution. The combining rule may be more straightforward and easier to understand than the corresponding aggregator, even though they may be equivalent.

4.8 OPEN UNIVERSE MODELS

The previously outlined work assumes that an agent knows which individuals exist and can identify them. It also assumed that individuals are not created or destroyed. Generally, we do not have to make these assumptions, see, e.g., Russell [2015] for a recent overview.

The problem of knowing whether two descriptions refer to the same individual is known as **identity uncertainty** [Pasula et al., 2003, Poole, 2007]. This arises in citation matching [Pasula et al., 2003] when we need to distinguish whether two references refer to the same paper and in record linkage [Fellegi and Sunter, 1969], where the aim is to determine if two hospital records refer to the same person (e.g., whether the current patient who is requesting drugs has been at the hospital before).

The problem of knowing whether some individual exists is known as **existence uncertainty** [Milch et al., 2005, Poole, 2007]. This is challenging because when existence is false, there is no individual to refer to, and when existence is true, there may be many individuals that fit a description. We may have to know which individual a description is referring to. In general, determining the probability of an observation requires knowing the protocol for how observations were made [Halpern, 2003].

Example 4.8 If an agent considers a house and declares that there is a green kitchen, the probability of this observation depends on what protocol they were using: did they report the color of the first room found, did they go looking for a green room, did they report the type of the first green thing found, or did they report on something that they thought was unusual?

Data that are reliable and people care about, particularly in the sciences, are being reported using combinations all of the issues of relational probabilistic modeling as well as the problems of describing the world at multiple levels of abstraction and detail, and handling multiple heterogenous data sets. It also requires new ways to think about **ontologies** [Poole et al., 2009], and new ways to think about the relationships between data, hypotheses and decisions.

4.8.1 IDENTITY UNCERTAINTY

To be able to count we need to be able to distinguish individuals.

Example 4.9 Suppose we are told that a room contains Sue's brother, Bill's son, Bill's father-in-law, and Sally's husband. From this description, we don't know how many people are in the room. There could be four people, if all of the descriptions refer to different people. It is possible that there are two people if some of the descriptions refer to the same person. It is even possible that there is only one person (if Bill's son married Bill's wife's mother) but that is not very common. To determine the distribution of the number of people, we need a model of why we were told

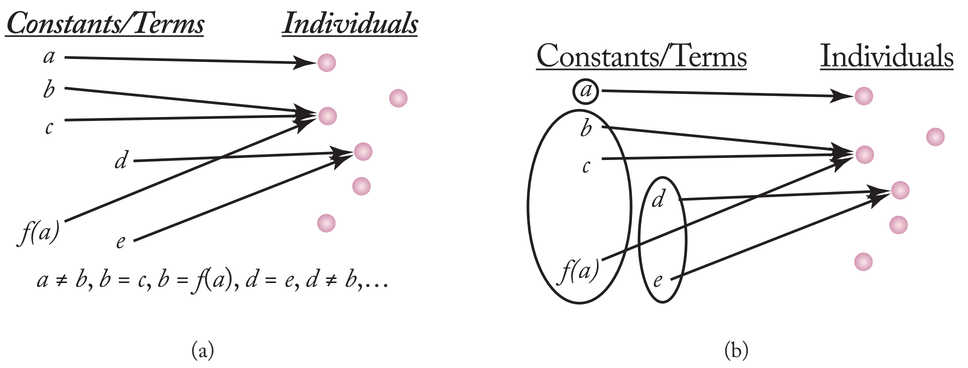


Figure 4.2: Equality of symbols.

this information and not other information. A robot may need to know the distribution of the number of people because, for example, it may need to bring in food.

In the standard semantics of first-order logic, there are individuals in the world, and terms in the language denote individuals in the world. We say that terms t_1 **equals** t_2, written $t_1 = t_2$, if t_1 and t_2 denote the same individual; see Fig. 4.2a. In terms of proofs, equality is reflexive, symmetric and transitive, and if $t_1 = t_2$ then t_1 can be substituted for t_2 without changing any truth values. This substitution property means than equality cannot be treated like any other binary relation.

Note that equality is very different from similarity. If $cup_1 = cup_2$, it is not the case that cup_1 is similar to cup_2; there is only one cup. If $cup_1 \neq cup_2$, the cups could be similar in all respects, but there are still two cups.

The problem of **identity uncertainty** or **equality uncertainty** is the problem of determining the probability that two descriptions denote the same individual.

A classic example of identity uncertainty is in **record linkage** [Fellegi and Sunter, 1969], the problem of determining if two records of people, objects, or events (e.g., hospital records or student records) denote the same individual or different individuals. If the two records denote the same individual, the differences between the records are errors. If the records denote different people, similarities are coincidences. Fellegi and Sunter [1969] treated it as a decision problem where the actions were to decide whether there was a link, non-link, or possible link (which can be reviewed by a person). Sharma and Poole [2005] built a probabilistic relational model based on this, and using similarity networks [Heckerman, 1990] to build graphical models of errors and coincidences (e.g., that people with the same last name often live together and twins have more features in common than would be expected by chance, and when people move they change many properties).

These models just considered pairwise matches (which may be appropriate if we are adding single people to a database that we assume is correct). For more general cases, due to the structure of equality (in particular, transitivity), pairwise matches do not scale. The other way to model equality is to realize that equality imposes a partition over terms; the elements of the same partition denote the same individual; see Fig. 4.2b. Thus, the probabilities of equality can be derived from a probability distribution over partitions. The number of partitions of n elements grows with the Bell number of n, which grows faster than any exponential. Thus it is impractical to have an explicit distribution over partitions.

A challenging application for identity uncertainty is in **citation matching**, such as is used in CiteSeerX [Wu et al., 2014]. The problem of determining which citations refer to the same paper is a problem of identity uncertainty. In a seminal work for citation matching, Pasula et al. [2003] used an MCMC algorithm for inducing the probability over partitions. Each partition of citations corresponds to a paper. The partitions can be merged and divided to generate the distribution over partitions. Wellner et al. [2004] used an undirected model for the same problem.

4.8.2 EXISTENCE UNCERTAINTY

In many cases we do not know whether an individual that fits a description actually exists. For example, see the following questions.

- What is the probability that there is a plane in this area given the observation of radar blip(s)?
- What is the probability that there is a large gold reserve in some region?
- What is the probability that this patient has a tumor?
- What is the probability that there is a third bathroom given there are two bedrooms?
- What is the probability that there are (exactly) three bathrooms given there are two bedrooms?

The problem of **existence uncertainty** is to determine whether some individual that fits a description actually exists. Modeling existence is challenging because non-existing objects do not have properties [Poole, 2007]. We need to be careful about exactly what doesn't exist when existence is false.

Consider the following problem due to Milch et al. [2005].

Example 4.10 Suppose there is radar system that is scanning the sky and reporting "blips" where it could have identified an airplane. The aim is to determine what airplanes exist and where they are. At each time, the radar reports location (other characteristics) of the blips it has detected. Note that this observation implicitly also implies that there were not blips at locations it was

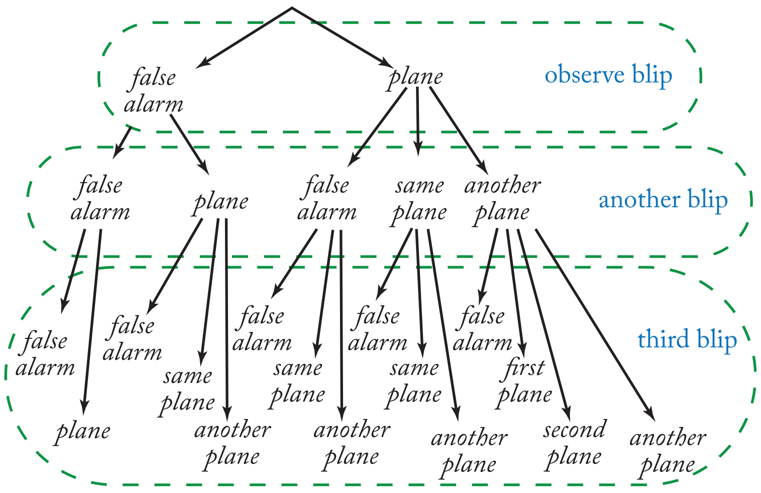


Figure 4.3: Observation of blips.

looking at. The aim is to get a distribution over the position and trajectories of planes that may exist.

Several solutions have been proposed in the literature; see also [Russell, 2015]. BLOG [Milch et al., 2005] is a generative model of existence, where first a number of objects are generated. For each number of objects, it models the correspondence of names to objects. For the above example, BLOG first generates a number of planes, and for each plane it generates a trajectory and other properties. This model then can determine the probability of the observations conditioned on.

Nonparametric Bayesian logic (NP-blog) [Carbonetto et al., 2005] is a generative model that generates the individuals sequentially, always asking whether given a new description, whether the description refers to an object it has already hypothesized, or whether there is a new (or different) object that has not been hypothesized yet.[1] So, NP-blog generates individuals as needed to explain observations. The process is shown in Fig. 4.3 for the example. When one blip (with its properties) is observed, it was either produced by a plane or it was a false alarm. If the plane exists, we can determine a distribution over its properties based on the properties of the blip. If there is another blip observed, then it is either a false alarm, was produced by another plane or was produced by the same plane that produced the first blip (if there was such a plane). Similarly, given a third blip, it was either produced by a plane known about, another plane or was a false alarm.

Poole [2008] gave the same example in the independent choice logic (similar to Problog). The representation follows the Blog model, where first the number of objects is generated. This

[1]Poole [2007] defined a probabilistic logic for existence and identity that is based on similar intuitions.

works by probabilistically creating a list of the objects that exist. Once we have a name for the object that exists, it can be endowed with properties that can be observed. This shows that we may not need to have a new logic to deal with existence uncertainty.

4.8.3 ONTOLOGIES

An **ontology** [Smith, 2003, Sowa, 2011, Janowicz et al., 2015] is a specification of the meaning of the terms used in an information system. More and more data sets are now being published with reference to formal ontologies. These are the sorts of data sets that statistical relation models make predictions for.

The interaction between ontologies and statistical relational models is interesting for a number of reasons.

- The information in the underlying ontologies is usually not stated in a data set. For example, a dataset would not contain information such as *mammal*(*sam*) if it also contained *person*(*sam*) and if the ontology specifies that all people are mammals. The implicit information is typically not stated, but can be inferred from the ontology. These inferred relationships may be the conditions required to make good generalizations; after all the reason why a term such as mammal was invented in the first place is because there are generalizations that occur for all mammals.

- It is reasonable to suggest that zero probabilities *only* arise from ontologies. In the robot kidnap problem a robot is moved to another position that it would be impossible to move to under its normal dynamics; however if this really has zero probability, it could not have happened. Between 1976 and 1985 the ozone hole over Antarctica grew at a rate that models deemed impossible, and so the only explanations was sensor failure, and so all of the sensor reading were rejected for many years. Both of these examples suggest that we should not have zero probabilities for anything that is possible, because zero probabilities cannot be updated based on more data. Just because something has never been observed does not mean it is impossible. However, ontological constraints which arise from definitions are necessary constraints and violations should have zero probability. For example, if a person is a mammal by definition, then any dataset that specifies that someone is a person but not a mammal should be rejected. It is then appropriate that the existence of a person who is not a mammal should have zero probability.

- There is often a mismatch between the ontologies that are common in the semantic web based on description logics (e.g., using OWL). Poole et al. [2009] has shown how these can be combined into a coherent framework based on what they call Aristotelian definitions, where each class is defined in terms of a genus (a superclass) and differentia (properties that distinguish this class from other subclasses of the genus). This allows classes to reduce to properties. Properties then can be mapped into relations and functions, which can have probabilities on them.

PART II

Inference

CHAPTER 5
Inference in Propositional Models

In order to prepare the stage for inference in relational probabilistic models, we first briefly review standard probabilistic and logical inference.

5.1 PROBABILISTIC INFERENCE

Probability has become a dominant paradigm in AI mainly because of the rise of graphical models, which make explicit conditional independence assumptions. The structure of these graphical models can be exploited for computational gain. To understand how (exact) inference algorithms can exploit the structure, consider the following example.

Example 5.1 Consider the belief network of Fig. 5.1. Suppose we observe $G = true$, which we write as g, and want the posterior probability of E. Thus we want, $P(E \mid g)$, which can be computed using:

$$P(E \mid g) = \frac{P(E \wedge g)}{\sum_E P(E \wedge g)} .$$

Thus, computing conditional probabilities can be reduced to computing the probability of conjunctions. To compute the probability of a conjunction, we add in all of the other variables, and sum them out in some order (called the **elimination ordering**). We do this because when we have all of the variables we can factorize the probability in terms of a product of factors. Thus, probabilistic inference reduces to computing a sum of products. To compute a sum over products we

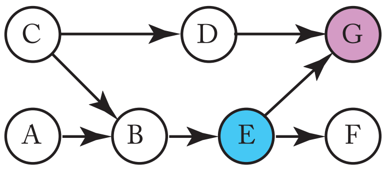


Figure 5.1: Belief network discussed in Example 5.1.

can use the distribution law to remove common factors from sums. In the example below, when summing out D, all of the factors that do not involve D can be distributed out of the sum, and similarly for the other variables, resulting in:

$$\begin{aligned}
P(E \wedge g) &= \sum_{ABCDF} P(ABCDEFg) \\
&= \sum_F \sum_B \sum_C \sum_A \sum_D P(A)P(B \mid AC)P(C)P(D \mid C) \\
&\qquad\qquad P(E \mid B)P(F \mid E)P(g \mid ED) \\
&= \left(\sum_F P(F \mid E)\right) \\
&\quad \sum_B P(E \mid B) \sum_C \left(P(C)\left(\sum_A P(A)P(B \mid AC)\right)\right. \\
&\qquad\qquad \left.\left(\sum_D P(D \mid C)P(g \mid ED)\right)\right) .
\end{aligned}$$

5.1.1 VARIABLE ELIMINATION

Variable elimination [Zhang and Poole, 1994] is a dynamic programming approach that exploits such a factorization by computing the innermost summations first, and storing the results. **Recursive conditioning** [Darwiche, 2001] is the search variant of variable elimination, which branches on the sums from the outside in. They do exactly the same additions and multiplications. Clique-tree propagation [Lauritzen and Spiegelhalter, 1988, Jensen et al., 1990, Shafer and Shenoy, 1990] uses the same factorization and compiles into a secondary structure that allows for computing the posterior marginal of all of the variables using only twice the time required to computing the marginal of one variable.

It does not matter whether the factors represent conditional probabilities or potentials in Markov models, the variable elimination algorithm is the same. We sum out variables from a product of factors. Directed models allow for (sometimes considerable) pruning before inference starts. In the example above, F can be pruned as we know that $\sum_F P(F \mid E)$ gives a factor of 1's.

5.1.2 RECURSIVE CONDITIONING

Here we present a variant of recursive conditioning for which the lifting—inference in the relational model without grounding all of the variables—is reasonably straightforward. The lifted version is presented later in Section 6.2.1.

In the variant of **recursive condition** presented in Fig. 5.2, factors are never modified; each one is evaluated when there is enough information to evaluate it. The context, *Con*, is a set of $X = v$ pairs, where X is a random variable and v is a value in the range of X. We say that the

```
procedure $rc(Con:$ context, $Fs:$ set of factors):
    if $\exists v$ such that $\langle\langle Con, Fs\rangle, v\rangle \in cache$
        return $v$
    else if $vars(Con) \not\subseteq vars(Fs)$
        return $rc(\{X = v \in Con : X \in vars(Fs)\}, Fs)$
    else if $\exists F \in Fs$ such that $vars(F) \subseteq vars(Con)$
        return $eval(F, Con) \times rc(Con, Fs \setminus \{F\})$
    else if $Fs = Fs_1 \uplus Fs_2$ where $vars(Fs_1) \cap vars(Fs_2) \subseteq vars(Con)$
        return $rc(Con, Fs_1) \times rc(Con, Fs_2)$
    else
        select variable $X \in vars(Fs)$
        $sum \leftarrow 0$
        for each $v \in range(X)$
            $sum \leftarrow sum + rc(Con \cup \{X = v\}, Fs)$
        $cache \leftarrow cache \cup \{\langle\langle Con, Fs\rangle, sum\rangle\}$
        return $sum$
```

Figure 5.2: Recursive conditioning with an explicit context.

variable X is assigned in Con if $X = v$ is in Con for some v. Fs is a set of factors defining the probabilistic model.

This computes $\sum_{\overline{X}} \prod_{F \in Fs} F \mid Con$ where $\overline{X}$ is the set of random variables in Fs that are not assigned in Con. This means that we are projecting all of the factors onto Con, multiplying the factors and summing out all of the variables not assigned in Con.

The first condition in Fig. 5.2 checks whether the answer has already been computed. Answers that have already been computed are stored in a cache. Initially the cache contains $\langle\langle\{\}, \{\}\rangle, 1\rangle$

$vars(E)$ returns the random variables in E, where E is a context or a set of factors. To ensure that the same case can be found in the cache, the second condition "forgets" variables in the context that are not in any factors. For example, the result of summing out D in the previous example can be used for all values of the variable B; when computing, e.g., $con(\{G = true, E = true, B = true, C = true\}, \{P(D \mid C), P(G \mid ED)\})$, the assignment to B is forgotten, so that after the first value is computed, the subsequent values can be retrieved from the cache.

The third condition checks whether a factor can be evaluated. A factor can be evaluated when all of its variables are assigned.

The fourth condition checks if the graph is disconnected given the assignment. This occurs when the FS can be partitioned into non-empty and disjoint subsets F_1 and F_2, where all variables in the intersection are assigned. In the previous example, when G, E, B, and C are assigned, the graph is disconnected and the factors involving A can be treated independently of the factors involving D.

The last condition branches on all of the value for one of variables. This algorithm sums over all of the values of the selected variable. The efficiency, but not the correctness, depends on which variable is selected.

The conditional probability $P(H = w \mid e)$, the probability of variable H given evidence e, can be computed using

$$P(H = w \mid e) = \frac{rc(\{H = w\} \cup e, Fs)}{(\sum_{v \in range(H)} rc(\{H = v\} \cup e, Fs)} \; .$$

All of these calls to rc can share the same cache.

The space and time complexity of this algorithm is $O(nr^t)$, for n variables, range size r, and treewidth t. The treewidth depends on the order the variables are selected.

5.1.3 BELIEF PROPAGATION

Finally, let us touch upon probabilistic inference via message-passing (on factor graphs). Probably the most prominent instance is the the **belief propagation** (**BP**) algorithm [Pearl, 1988]. It is exact when the factor graph is a tree, and does the same additions and multiplications as variable elimination and recursive conditioning (with appropriate elimination orderings) for these cases. When the factor graph has cycles and is not a tree, it is only approximate.

Once evidence has been incorporated into the model by setting $f(\mathbf{x}) = 0$ for states $\mathbf{x}$ that are incompatible with the evidence, the BP algorithm sends messages between variable nodes and their neighboring factors (and vice versa) until convergence. Specifically, the message from a variable X to a factor f is

$$\mu_{X \to f}(x) = \prod\nolimits_{h \in \mathrm{nb}(X) \setminus \{f\}} \mu_{h \to X}(x),$$

where $\mathrm{nb}(X)$ is the set of factors X appears in. The message from a factor to a variable is

$$\mu_{f \to X}(x) = \sum\nolimits_{\neg\{X\}} \left(f(\mathbf{x}) \prod\nolimits_{Y \in \mathrm{nb}(f) \setminus \{X\}} \mu_{Y \to f}(y) \right) \; ,$$

where $\mathrm{nb}(f)$ are the arguments of f, and the sum is over all of these except X, denoted as $\neg\{X\}$. Initially, the messages are set to 1. After convergence or a predefined number of iterations, the unnormalized belief of each variable X_i can be computed from the equation $b_i(x_i) = \prod_{f \in \mathrm{nb}(X_i)} \mu_{f \to X_i}(x_i)$. As already mentioned, these computed marginal probability functions will be exact if the factor graph has no cycles, but the BP algorithm is still well-defined when the factor graph does have cycles. Although this loopy BP has no guarantees of convergence or of giving the correct result, in practice it is often close, and can be much more efficient than other methods [Murphy et al., 1999].

5.2 LOGICAL INFERENCE

Logic is the other dominant paradigm in AI mainly because it provides a precise specification of tasks, independent of how results are computed, and logical structure can be exploited for computational gain. In this section, we show how to compute models for logical theories. We also show how the basic solvers can be extended for use in probabilistic reasoning, which will involve moving from a pure satifiability approach to one that is based on model counting. We will first cover propositional logic, and then discuss how it can be lifted toward logic programs.

5.2.1 PROPOSITIONAL LOGIC, SATISFIABILITY, AND WEIGHTED MODEL COUNTING

Recall (from Equation 2.3, page 23) that a propositional clause is of the form

$$A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m \ .$$

Assume that we are given a set of propositional clauses. The problem of determining if a model exists for such a theory is the famous **satisfiability problem** (**SAT**). That is, the **SAT** problem consists of deciding whether there is an interpretation I that satisfies all clauses in the theory, where a clause

$$A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m \ ,$$

is satisfied by an interpretation I if one B_i is false in I or one A_i i true in I. A standard algorithm for deciding satisfiability is the **David-Putnam-Logeman-Loveland** (DPLL) procedure shown in Algorithm 5.3, which is closely related to recursive conditioning. In the algorithm we use the notation $\bar{l}$ to denote the complement of the literal l, that is, if l is a positive atom, then $\bar{l} = \neg l$, otherwise the literal $l = \neg a$ is a negated atom and $\overline{\neg a} = a$.

The basic SAT problem can be extended in various ways. For instance, **#SAT** is the problem of deciding how many models a theory has. Note that for #SAT, we must specify the clauses and the atoms, because it is possible to have an atom without any clauses; adding an atom without any clauses doubles the number of interpretations.

Weighted model counting (WMC), is a step toward closing the gap between probabilistic and logical inference. In weighted model counting, each literal ℓ (positive and negative) is given a non-negative weight $w(\ell)$. The weight of an interpretation I, written $w(I)$, is the product of the weights of the literals that are true in the interpretation, i.e., $w(I) = \prod_{I \models \ell} w(\ell)$. The weighted model count $WMC(Th)$ of a theory Th is the sum of the weights of all its models. That is,

$$\begin{aligned} WMC(Th) &= \sum_{I \models Th} w(I) \\ &= \sum_{I \models Th} \prod_{I \models \ell} w(\ell) \ , \end{aligned}$$

```
procedure $DPLL(Th :$ set of clauses):
    if $Th$ is empty
        return $true$
    else if $Th$ contains an empty clause
        return $false$
    else if there exists a literal $l$ in $Th$ such that $\bar{l}$ does not appear in $Th$
        $Th' := Th - \{C | C \in Th$ and $C$ contains $l\}$
        return DPLL($Th'$)
    else if $Th$ contains a unit clause (a clause with one literal $l$)
        $Th' := \{C - \{\bar{l}\} | C \in Th$ and $C$ does not contain $l\}$
        return DPLL($Th'$)
    else choose a proposition $l$ in $Th$
        $Th_t := \{C - \{\bar{l}\} | C \in Th$ and $C$ does not contain $l\}$
        $Th_f := \{C - \{l\} | C \in Th$ and $C$ does not contain $\bar{l}\}$
        return $DPLL(Th_t) \vee DPLL(Th_f)$
```

Figure 5.3: The Davis-Putnam-Logemann-Loveland Procedure.

#SAT is the unweighted version of WMC, where every literal has a weight of 1.

Example 5.2 Consider the theory Th consisting of the clause:

$$burglary \leftarrow alarm .$$

Assume the weights $w(alarm) = 0.05$, $w(\neg alarm) = 0.95$, $w(burglary) = 1$, $w(\neg burglary) = 8$. Then there are 3 models: $\{alarm, burglary\}$ with weight = 0.05×1; $\{\neg alarm, burglary\}$ with weight = 0.95×1; and $\{\neg alarm, \neg burglary\}$ with weight = 0.95×8. Thus $WMC(Th)$ returns 8.6.

The SAT algorithm can be adapted to solve the WMC problem; the resulting algorithm of Birnbaum and Lozinskii [1999] is shown in Algorithm 5.4.

The algorithm keeps track of the set of atoms. To see why it needs to, consider the case where all weights are equal to 1, and there are no clauses If there are n different atoms, there are 2^n models. This is the number that will be returned in the first if-test of the algorithm. In the general WMC case, the weights will not be equal to 1, which explains why WCM returns $\prod_{a \in A}(w(a) + w(\neg a))$ in this case.

If a literal l must be true in a theory, this will yield a factor $w(l)$ in the WMC to all models of the theory, which explains the use of w in that last two cases in the algorithm.

5.2.2 SEMIRING INFERENCE

While recursive conditioning and DPLL, may look quite different they are both instances of a more general search algorithms for **commutative semirings**. Understanding this connection is

```
procedure $WMC(Th:$ set of clauses, $w:$ weight function $, A:$ set of atoms):
    if $Th$ is empty
        return $\prod_{a \in A}(w(a) + w(\neg a))$
    else if $Th$ contains an empty clause
        return 0
    else if $Th$ contains a unit clause (a clause with one literal $l$)
        $Th' := \{C - \{\bar{l}\} | C \in Theory$ and $C$ does not contain $l\}$
        let $A'$ be $A$ with the atom in $l$ removed
        return $w(l) \times WMC(Th', w, A')$
    else choose a proposition $l$ in $Th$
        $Th_t := \{C - \{\bar{l}\} | C \in Theory$ and $C$ does not contain $l\}$
        $Th_f := \{C - \{l\} | C \in Theory$ and $C$ does not contain $\bar{l}\}$
        let $A'$ be $A$ with the atom in $l$ removed
        return $w(l) \times WMC(Th_t, w, A') + w(\bar{l}) \times WMC(Th_f, w, A')$
```

Figure 5.4: Search algorithm for weighted model counting.

important for understanding how to exploit both the graphical structure of graphical models as well as the logical structure of logic programs.

A commutative semiring has two associative and commutative operations $\oplus$ and $\otimes$, where $\otimes$ distributes over $\oplus$ (i.e., $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$). Given a set v of variables, and a set f of factors, where a factor is a function on a subset of variables, the aim is to compute $\oplus_x \otimes_{f \in F} f(x)$. Various problems can be put into this abstraction as in the following table, where unit is the value u such that $x \otimes u = x$:

problem	Variables	Factors	$\oplus$	$\otimes$	unit
probabilistic inference / WMC	RVs.	CPDs/potentials	$\sum$	$\prod$	1
SAT	propositions	clauses	$\vee$	$\wedge$	true
optimization	variables	factors	max	$\sum$	0
database query	attributes	table	union	join	Ø

Figure 5.5 gives a generic search algorithm to to compute $\oplus_x \otimes_{f \in F} f(x)$, assuming Boolean variables. It can easily be extended to non-Boolean variables. For each particular instance of the algorithm, there are special operations that can be carried out for assigning values to variables. For example assigning a variable to a value in the database, selects the value for that variable and then projects onto the other variables. In SAT, assigning a value to a variable that appears in a clause either removes the clause or removes the variable.

In the SAT case, clauses can be evaluated when one of the disjuncts is true or they are all false. SAT also exploits the fact that the logical connectives can be short-circuited, e.g., *false* $\wedge\ x$ is *false* and we do not need to evaluate x, and *true* $\vee\ x$ is *true*.

```
procedure $SS(Factors, Variables)$:
    if $Factors$ is empty
        return unit
    else if $f \in Factors$ can be evaluated
        return $f \otimes SS(Factors \setminus \{f\}, Variables)$
    else select a variable $v$ in $Variables$
        $Factors_t$ := $Factors$ with $v$ assigned to $true$
        $Factors_f$ := $Factors$ with $v$ assigned to $false$
        return $SS(Factors_t, Variabes \setminus \{v\}) \oplus SS(Factors_f, Variabes \setminus \{v\})$
```

Figure 5.5: Abstract semiring search for Boolean variables.

Recursive conditioning adds **caching** and recognizing disjoint components so that it can exploit the independencies of a graphical model. Within SAT, **clause learning** is probably the most successful form of caching.

This abstraction has been known for a long time, e.g., Lauritzen and Spiegelhalter [1988] make explicit the mapping between the probabilistic inference and the database notions.

5.2.3 THE LEAST HERBRAND MODEL

While clausal theories can have no, one, or more than one model, the semantics of a logic programs is given by its **least Herbrand model**. This least Herbrand model can be defined (and computed with) the T_P operator, which we now define for propositional logic programs, that is, a set of clauses of the form

$$A \leftarrow B_1 \wedge \cdots \wedge B_m \ .$$

The T_P operator is defined for a logic program P and a set of propositions I as follows:

$$T_P(I) = I \cup \{A | (A \leftarrow B_1 \wedge \cdots \wedge B_m) \in P \text{ and } \{B_1, \ldots, B_m\} \subseteq I\} \ . \tag{5.1}$$

The least Herbrand model $\mathrm{LH}(P) = T_P^\infty(\emptyset)$ is now defined by applying the T_P operator on the empty set and then repeatedly applying the T_P operator on the result until a fix point is reached, that is, until $T_P(I) = I$.

Example 5.3 Consider, for instance, the following program:

$$\begin{array}{rcl} flies & \leftarrow & bird, normal. \\ bird & \leftarrow & tweety. \\ bird & \leftarrow & oliver. \\ normal & \leftarrow & tweety. \\ tweety\ . & & \end{array}$$

This will result in the following sequence of models

$$\begin{array}{lcl} T_P(\emptyset) & = & \{tweety\} \\ T_P^2(\emptyset) & = & \{tweety, normal, bird\} \\ T_P^3(\emptyset) & = & \{tweety, normal, bird, flies\} \\ T_P^4(\emptyset) & = & T_P^3(\emptyset) \ . \end{array}$$

Given that logic programs are also clauses, the question arises as to whether it is possible to simply use a SAT solver to compute the least Herbrand model instead of resorting to the T_P operator. The answer to this question is that this is possible in general by rewriting the logic program.

A logic program is acyclic if no atom depends on itself (i.e., atom a does not appear in a proof for a). For **acyclic logic programs** (with negation as failure[1]), one can use **Clark's completion**. While a definite clause specifies sufficient conditions for the conclusion to hold, Clark's completion assumes that the conjunction of all sufficient conditions for a particular predicate are also necessary conditions.

Example 5.4 The completed program of *flies* is listed below. Notice that the sufficient conditions for *bird* are *tweety* or *oliver* and that Clark's completion now rewrites this as an equivalence rather than an implication. It can easily be verified that the only model of the resulting theory coincides with the least Herbrand model of the original logic program.

$$\begin{array}{rcl} flies & \leftrightarrow & bird \wedge normal. \\ bird & \leftrightarrow & tweety \vee oliver. \\ normal & \leftrightarrow & tweety. \\ tweety. & & \\ oliver & \leftrightarrow & false \ . \end{array}$$

This uniqueness of the model of the completion does not hold for **cyclic programs**, that is, programs in which propositions depend on themselves. There are three cases that can be exemplified by simple examples.

- The clauses $a \leftarrow b$ and $b \leftarrow a$ have as a completion $a \leftrightarrow b$, which has two models, one where both are true, and one where both are false. The minimum model has both being false. This sort of cyclicity is allowed in Problog, and was used in Example 3.12. More complex transformations of logic programs guarantee in such cases the correspondence between the least Herbrand model and the model of the transformed program [Janhunen, 2004].

[1]This means negation in the body of a clause, and negation means to prove $\neg p$ from failure to prove p. Without negation, the rewriting can be simplified.

- The clauses $a \leftarrow \neg b$ and $b \leftarrow \neg a$ have as a completion $(a \leftrightarrow \neg b) \wedge (b \leftrightarrow \neg a)$, which has two models, $\{a, \neg b\}$ and $\{b, \neg a\}$. Symmetry implies that neither would be preferred to the other. These are both stable models, and form answer sets. P-Log [Baral et al., 2004, Baral and Hunsaker, 2007] defined a distribution over these answer sets.

- The clauses $a \leftarrow b$ and $b \leftarrow \neg a$ have as a completion $(a \leftrightarrow b) \wedge (b \leftrightarrow \neg a)$, which is logically inconsistent and so has no models. For the probabilistic case, Poole and Crowley [2013] advocated using the equilibrium distribution of the induced Markov chain.

5.2.4 GROUNDING

So far, we have focused on propositional logic, but we are actually interested in relations and the predicates calculus, which requires dealing with constants, terms, and variables.

One way of dealing with first-order logic is by **grounding** all formulae so that one obtains a set of formulae without any variables. The resulting formulae can then be interpreted as propositional formulae by simply intepreting each ground fact as a propositional predicate. This also implies that on the resulting theory one can invoke a SAT solver.

Example 5.5 Reconsider our *human* theory from Example 2.2 and suppose there are two constants *ann* and *paul*. By applying all possible substitutions to the clauses in that theory we obtain the grounded theory:

$$\begin{array}{rcl}
human(paul) & \leftarrow & male(paul). \\
human(paul) & \leftarrow & female(paul). \\
human(ann) & \leftarrow & female(ann). \\
human(ann) & \leftarrow & male(ann). \\
false & \leftarrow & male(ann) \wedge female(ann). \\
false & \leftarrow & male(paul) \wedge female(paul)\ .
\end{array}$$

By applying a SAT solver one can compute a model for this theory, which could result in one of the models listed in Example 2.6.

While grounding is a basic theoretical construct used to define the semantics, it should be clear that from a practical perspective there are some key concerns. First, the number of groundings is infinite whenever functors are used, cf. the *nat* program of Fig. 2.4. Second, even when there are no functors, the number of groundings grows exponentially with the number of variables in the formulae. If there are n constants in the Herbrand domain, grounding a clause with k variables results in n^k possible groundings.

One wants to ground out only those parts that are really needed. Unfortunately, it is impossible to determine what grounding may be needed for a logic program with function symbols, because the language is Turing equivalent, and a fixed finite grounding gives a decidable ground model.

5.2.5 PROVING

The above methods were used to find a model. The opposite problem is to decide if some proposition a is a logical consequence of the clauses, C, which can be done by showing that $C \cup \neg a$ is inconsistent (does not have a model). Note that for definite clauses, finding a *minimal* model is equivalent to finding proofs, because each atom in a minimal model is a logical consequence of the definite clauses.

Proofs are typically constructed using a **resolution** procedure, which we explain here only for logic programs. The resulting **SLD-resolution procedure** works as follows: given a **goal** $G_1, G_2 \ldots, G_n$, with free variables $\overline{x}$, create the answer clause $yes(\overline{x})$:-$\mathtt{G_1}, \mathtt{G_2} \ldots, \mathtt{G_n}$. Given an answer clause y:-$G_1, G_2 \ldots, G_n$ and a clause G:-$L_1, \ldots, L_m$ in the logic program such that $G_1\theta = G\theta$, applying SLD resolution yields the new answer clause $y\theta$:-$L_1\theta, \ldots, L_m\theta, G_2\theta \ldots, G_n\theta$. A *successful* proof of a goal is then a sequence of resolution steps yielding the empty answer clause, i.e., y:- . *Failed* proofs do not end in the empty goal.

Example 5.6 The query, $grandparent(X, ann)$, representing the question "who is Ann's grandparent" has the following SLD proof:

$$\begin{array}{l} yes(X)\text{:-}grandparent(X, ann) \\ yes(X)\text{:-}parent(X, Z), parent(Z, ann) \\ yes(jeff)\text{:-}parent(paul, ann) \\ yes(jeff)\text{:-}\ \ . \end{array}$$

Which indicates that one answer is $X = jeff$, so $grandparent(jeff, ann)$ is a logical consequence of the logic program.

Resolution is employed by many theorem provers (such as Prolog). Indeed, when given the goal $grandparent(X, ann)$, Prolog would compute the above successful resolution refutation and answer that the goal is true.

CHAPTER 6

Inference in Relational Probabilistic Models

In this chapter, we touch upon the problem of performing inference in relational probabilistic models. Inference in probabilistic relational models refers to computing the posterior distribution of some random variables given some evidence. There are many ways of doing inference. Conceptually the easiest one is "**grounded inference**."

6.1 GROUNDED INFERENCE FOR RELATIONAL PROBABILISTIC MODELS

A standard way to carry out inference in probabilistic logical models is to (try to) generate and ground as few of the random variables as possible. In undirected models, this typically means considering the whole grounding. Directed models allow pruning of irrelevant random variables or only considering the relevant random variables (e.g., in ICL and ProbLog, the relevant ground instances can be carried out using a form of abduction [Poole, 1993a]). Once the model is grounded, standard inference methods (such as those studied in the previous chapter) can be applied to generate answers to the probabilistic queries.

For the parameterized probabilistic models of Chapter 3, we have already discussed grounding and it should be clear that standard probabilistic inference techniques (such as those seen in the previous chapter) directly apply to the resulting groundings. In recent years, however, an alternative and fairly general way of performing probabilistic inference has become popular. It is based on grounding and a reduction to **weighted model counting**.

6.1.1 WEIGHTED MODEL COUNTING

One attractive avenue for probabilistic inference is to turn it into a weighted model counting (WMC) problem. We discuss how basic inference methods for Markov Logic and ProbLog can exploit WMC.

6.1.2 WMC FOR MARKOV LOGIC

For MLNs, we ground the MLN at hand and transform it to a WMC problem [Van den Broeck, 2011] as follows. For every formula $w : f$ in the MLN, ground it in every possible way, yielding a set of formulae $w : f\theta$. Then, as discussed in Section 4.6, for each such formula a unique predicate

$n\theta$ is constructed and the formula $n\theta \leftrightarrow f\theta$ is added together with $w(n\theta) = e^w$ and $w(\neg n\theta) = 1$ to the WMC problem. The weights for all other literals are set to 1. The idea is that the literal $n\theta$ is true exactly when the soft constraint $w : f$ is satisfied for substition θ.

Example 6.1 Reconsider the first two rules of the smokers Example 3.8:

$$\begin{array}{ll} 1.5: & cancer(P) \leftarrow smoking(P) \\ 1.1: & smoking(X) \leftarrow friends(X,Y) \wedge smoking(Y) \end{array}$$

and assume we have two constants a and b. Then we would obtain the following WMC problem:

$$\begin{array}{rcl} r1(a) & \leftrightarrow & (cancer(a) \leftarrow smoking(a)) \\ r2(b) & \leftrightarrow & (cancer(b) \leftarrow smoking(b)) \\ r3(a,a) & \leftrightarrow & (smoking(a) \leftarrow friends(a,a) \wedge smoking(a)) \\ r4(a,b) & \leftrightarrow & (smoking(a) \leftarrow friends(a,b) \wedge smoking(b)) \\ r5(b,a) & \leftrightarrow & (smoking(b) \leftarrow friends(b,a) \wedge smoking(a)) \\ r6(b,b) & \leftrightarrow & (smoking(b) \leftarrow friends(b,b) \wedge smoking(b)) \end{array}$$

with weights $w(r1(a)) = w(r2(b)) = e^{1.5}$, $w(r3(a,a)) = w(r4(a,b)) = w(r5(b,a)) = w(r6(b,b)) = e^{1.1}$, and the weight of all other literals equal to 1. This logical formulae can then easily be transformed into clausal form and used by a WMC solver.

6.1.3 WMC FOR PROBLOG

The conversion for non-cyclic ProbLog programs to WMC goes as follows, see e.g., Fierens et al. [2015]. It first requires grounding the relavant parts of the program, then converting it according to Clark's completion, and then setting the weights. For each ground probabilistic fact (atomic choice) $p :: f\theta$, we set $w(f\theta) = p$ and $w(\neg f\theta) = 1 - p$, and set the weights of all other literals to 1.

Example 6.2 Consider the following variant of the *alarm* network and one constant *john*:

$$\begin{array}{rcl} & & 0.01 :: burglary. \\ & & 0.05 :: earthquake. \\ & & 0.7 :: hearsalarm(X). \\ alarm & \leftarrow & burglary. \\ alarm & \leftarrow & earthquake. \\ calls(X) & \leftarrow & alarm, hearsalarm(X)\ . \end{array}$$

Then we would obtain the theory and weights:

$$\begin{aligned}
&\mathit{alarm} \leftrightarrow \mathit{burglary} \vee \mathit{earthquake}. \\
&\mathit{calls}(\mathit{john}) \leftrightarrow \mathit{alarm} \wedge \mathit{hearsalarm}(\mathit{john}). \\
&w(\mathit{burglary}) = 0.01, w(\neg\mathit{burglary}) = 0.99, \\
&w(\mathit{earthquake}) = 0.05, w(\neg\mathit{earthquake}) = 0.95, \\
&w(\mathit{hearsalarm}(\mathit{john})) = 0.7, w(\neg\mathit{hearsalarm}(\mathit{john})) = 0.3 \ .
\end{aligned}$$

As mentioned in Section 5.2.3, cyclic programs require a more complex transformation such as that by Janhunen [2004] to guarantee that the resulting set of clauses captures the least Herbrand model.

In practice, in formalisms such as ICL and ProbLog, one need not ground the complete program, but can use a pre-processing step in which only those clauses that are necessary for answering a query are computed. This can be implemented using SLD-resolution with abduction to find the relevant rules and weights [Poole, 1993a]. For instance, if one is only interested in the marginal probability of *alarm*, the rules for *hearsalarm*(X) and *calls*(X) are not needed.

6.1.4 KNOWLEDGE COMPILATION

One technique that is often used in Bayesian networks, probabilistic logic programming, and probabilistic databases is to use **knowledge compilation** to compile logical formulae into a representation that allows for tractable inference. This is used, for instance, by the ProbLog implementations [Fierens et al., 2015], where the logical formula of a WMC problem is compiled into a d-DNNF, that is, into deterministic, decomposable negation normal form, which then allows to efficiently compute the WMC.

Example 6.3 (Adapted from Fierens et al. [2015].) Continuing the above example, assume that we are given the logical formula:

$$\begin{aligned}
&\mathit{alarm} \leftrightarrow \mathit{burglary} \vee \mathit{earthquake} \\
&\mathit{calls}(\mathit{john}) \leftrightarrow \mathit{alarm} \wedge \mathit{hearsalarm}(\mathit{john}) \\
&\mathit{calls}(\mathit{john})
\end{aligned}$$

where the last fact could represent the evidence for a query.[1] The d-DNNF for this formula is shown in Fig. 6.1.

A negation normal form formula (NNF) is a rooted directed acyclic graph in which each leaf node is labeled with a literal and each internal node is labeled with a conjunction or disjunction. A decomposable negation normal form (DNNF) is a NNF satisfying decomposability: for every conjunction node, it should hold that no two children of the node share any atom with each

[1]Note that this is not the same as adding the evidence as a fact, because that would affect the completion. The evidence needs to be added to the completed clauses.

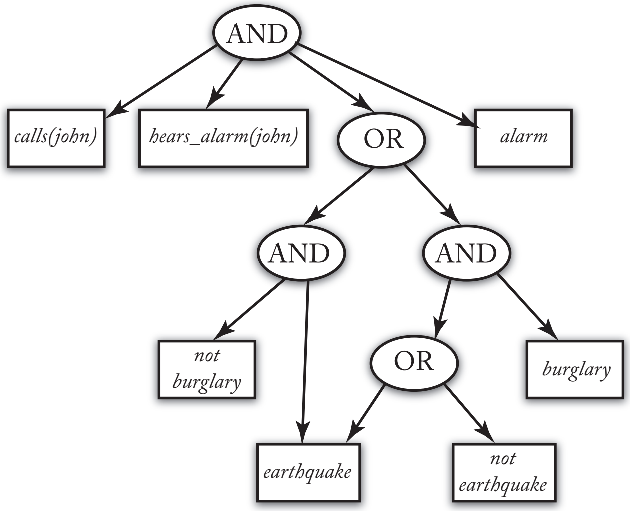


Figure 6.1: A logical formula in d-DNNF form (from Fierens et al. [2015]).

other. A deterministic DNNF (d-DNNF) is a DNNF satisfying determinism: for every disjunction node, all children should represent formulas that are logically inconsistent with each other. Decomposability allows one to compute the probability of the conjuncts as their products, while determinism allows one to sum the probabilities of the disjuncts. For WMC, we need a d-DNNF that also satisfies smoothness: for every disjunction node, all children should use exactly the same set of atoms. Compiling a Boolean formula to a (smooth) d-DNNF is a well-studied problem, and several compilers are available, e.g., Darwiche [2004]. The d-DNNF's are then typically turned into arithmetic circuits for inference and learning, cf. Darwiche [2009].

It should be mentioned that d-DNNF is just one representation that is used as a target language for knowledge compilation. Many other representations exists that differ in the operations they support efficiently [Darwiche and Marquis, 2002]. Some of these, like d-DNNFs, have also been extended for use with lifted inference, cf. Van den Broeck et al. [2011].

6.2 LIFTED INFERENCE: EXPLOITING SYMMETRIES

While grounding to a propositional representation may be an intuitive way to define the semantics and to perform inference, we can do better by exploiting the symmetry and regularity inherent in the relational structure. Such regularities and symmetries can often be easily modeled using probabilistic logical models. These models allow us to encode large, complex models using few rules only and, hence, symmetries and redundancies abound. Inference by grounding at the propositional representation level does not exploit the symmetries.

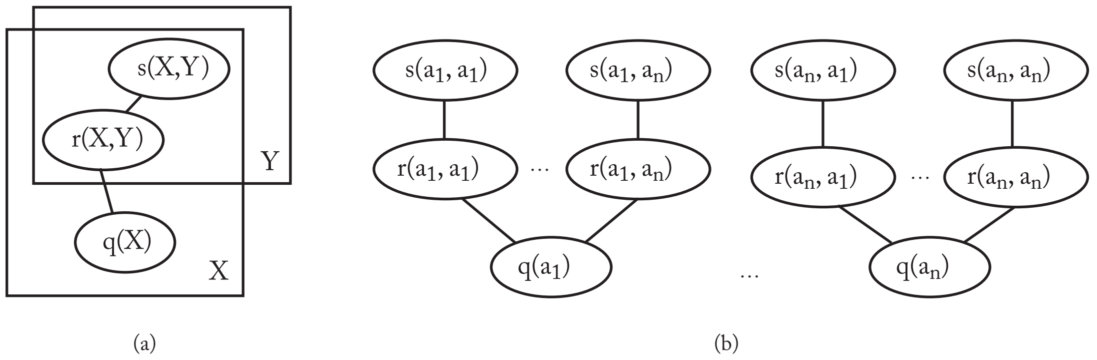


Figure 6.2: A parameterized graphical model (a) and its grounding (b).

More recently, there has been work on **lifted probabilistic inference** [Poole, 2003, de Salvo Braz et al., 2007, Singla and Domingos, 2008, Milch et al., 2008], where the idea is to carry out probabilistic reasoning at the lifted level, without grounding out the parameterized random variables. Instead, we count how many of the probabilities we need, and when we need to multiply a number of identical probabilities, we can take the probability to the power of the number of individuals, as in the following example.

Example 6.4 Consider the parameterized graphical model of Fig. 6.2(a) with two parameterized factors, $\langle\{\}, \{s(X,Y), r(X,Y)\}, \phi_1\rangle$ and $\langle\{\}, \{r(X,Y), q(X)\}, \phi_2\rangle$. Suppose the population for X and Y is $\{a_1, \ldots, a_n\}$. In the grounding, shown in Fig. 6.2 (b), there are n disconnected components. Each component has $2n + 1$ random variables and $2n$ factors.

Suppose we wanted to sum out all of the random variables, to compute the partition function. Let's first sum out the n^2 random variables that are instances of $s(X,Y)$. This can be done in one step at the lifted level, creating a parfactor

$$\langle\{\}, \{r(X,Y)\}, \phi_3\rangle,$$

where ϕ_3 is $\sum_S \phi_1$, obtained by summing out $s(X,Y)$ from ϕ_1.

Suppose now we want to sum out $r(X,Y)$. In the grounding, for every X, there are n identical factors which need to multiplied. The resulting factors can be represented as:

$$\langle\{\}, \{q(X)\}, \phi_4\rangle$$

where ϕ_4 is $(\sum_r \phi_2\phi_3)^n$, where taking a factor to the power n means taking each value to the power n. The partition function is then $(\sum_q \phi_4)^n$. This example shows the potential of lifted inference in exploiting the symmetry implied by exchangeability.

The problem of **lifted probabilistic inference** was first explicitly proposed by Poole [2003], who formulated the problem in terms of parameterized random variables. He also introduced the use of splitting to complement unification, the parfactor representation of intermediate results, and an algorithm for summing out parametrized random variables and multiplying factors in a lifted manner. It could solve Example 6.4 in a lifted manner. However, not all cases can be solved with these techniques.

Example 6.5 Consider eliminating $q(X)$ first in Example 6.4. The instances of X are independent, and so this can be carried out in a lifted manner with respect to X. For each X, eliminating $q(X)$ constructs a factor on $\{r(X, a_1), \ldots, r(X, a_n)\}$. De Salvo Braz et al. [2005, 2007] realized that in such cases, rather than representing the resulting factor, we only need the count of the number of instances of $r(X, Y)$ which have a certain value. If r is binary, we only need to consider the cases where r is true for i individuals and is false for $n - i$ individuals, for $i \in [0, n]$. If $r(X, Y)$ has k values, there are $O(n^{k-1})$ cases that need to be considered.

Counting elimination [de Salvo Braz et al., 2005, de Salvo Braz et al., 2007] handles cases where grounding followed by eliminating creates factors containing random variables for all individuals in a population. In many such cases, it is only the counts—the histogram of the number of instances for each combination of individuals—that characterize the factor. Thus, instead of creating a factor exponential in the population, counting elimination does a case analysis of a polynomial number of cases. Milch et al. [2008] proposed counting formulae as a representation of the intermediate result of counting, which allowed for more cases where counting was applicable. This work has been advanced by Taghipour and Davis [2012], Taghipour et al. [2013] to include more general constraints in the parfactors and by Mittal et al. [2014] to include novel lifting rules resulting in even smaller models. These algorithms are complicated and they still need to ground in many cases.

The main problem is that the proposals are based on variable elimination [Zhang and Poole, 1994]. This is a dynamic programming approach which requires a representation of the intermediate results, and the current representations for such results are not closed under all of the operations used for inference.

Some of the more recent exact algorithms are search based [Gogate and Domingos, 2010, 2011, Jha et al., 2010, Van den Broeck et al., 2011, Poole et al., 2011]. These methods are promising because the search only assigns values and simplifies the factors. For instance, Gogate and Domingos [2011] reduced the problem of lifted probabilistic inference to weighted model counting in a lifted graph. Given a relational probabilistic model, the resulting **probabilistic theorem proving** decides upon a lifted inference rule to apply (conditioning, decomposition, partial grounding, etc.), constructs a set of reduced relational probabilistic models, recursively calls itself on each reduced MLN in this set, and combines the returned values in an appropriate manner. Another recent approach to lifted inference, called **first-order knowledge compilation**, compiles the lifted graph into a weighted CNF (see also Section 6.1.1), and then runs weighted model counting on the compiled network [Van den Broeck et al., 2011, 2014].

Lifted inference turns out to be a very difficult problem, as the possible interactions between parameterized random variables can be very complicated. None of the above methods work for all cases, meaning that they may have to resort to "ground" inference. Jaeger [2000] was the first to show the limitations of lifted inference for reasonably rich languages (which include quantification and equality). Recently, Jaeger and Van Den Broeck [2012] showned the limitations for representations that include binary predicates, and are only slightly more complex than shown in Fig. 6.2. Observations can make such examples intractable. Van den Broeck [2011] started to study the completeness of lifted inference, and Gribkoff et al. [2014], Beame et al. [2015] recently analyzed the complexity of the (weighted) first-order model counting problem.

6.2.1 EXACT LIFTED INFERENCE

Here we give a basic algorithm for **lifted recursive conditioning**. This lifts the recursive conditioning algorithm of Fig. 5.2. Note that we chose that algorithm, which separates the context and the factors, because it makes it easier to explain. Furthermore, many other existing lifting inference methods can be viewed as variants of the algorithm.

In recursive conditioning (page 66), a context is a set of assignments of values to variables. Here we extend a context to also include counts, where a **count** is of the form $\#_X\phi(X) = n$, where ϕ is a conjunction of PRVs. each with only X free, which means that there are n instances of X for which formula ϕ is true of X.

The algorithm implements $lrc(Con, Fs)$ where:

- Con is a set consisting of counts and assignments to random variables and
- Fs is a set of weighted formulae of the form $\langle formula, weight\rangle$.

A weighted formula can be evaluated in context Con if Con specifies how many instances of the weighted formula hold. Let $eval(F, Con)$ be the value that weighted formula F evaluates to in context Con. We show the evaluation in a series of examples.

Example 6.6 Consider the context:

$$\begin{aligned} Con_1 = \{\neg A,\ & \#_X f(X) \wedge g(X) = 7, \\ & \#_X f(X) \wedge \neg g(X) = 5, \\ & \#_X \neg f(X) \wedge g(X) = 18, \\ & \#_X \neg f(X) \wedge \neg g(X) = 0\} \end{aligned}$$

and the weighted formulas:

$$\begin{aligned} Fs_1 = \{ & \langle \neg a \wedge \neg f(X) \wedge g(X), 0.1\rangle, \\ & \langle a \wedge \neg f(X) \wedge g(X), 0.2\rangle, \\ & \langle f(X), 0.25\rangle, \\ & \langle X \neq Y \wedge f(X) \wedge g(Y), 0.3\rangle, \\ & \langle f(X) \wedge f(Y), 0.35\rangle, \\ & \langle f(X) \wedge h(X), 0.4\rangle\} . \end{aligned}$$

Given Con_1, there are 18 instances of $\neg a \wedge \neg f(X) \wedge g(X)$ that are true in the grounding, so the first weighted formula in Fs_1 can be evaluated:

$$eval(\langle \neg a \wedge \neg f(X) \wedge g(X), 0.1\rangle, Con_1) = 0.1^{18} .$$

There are no instances of the second weighted formula in Con_1 as a is false in Con_1, so

$$eval(\langle a \wedge \neg f(X) \wedge \neg g(X), 0.2\rangle, Con_1) = 1$$

and so this weighted formula is ignored in this context. There are 12 instances of $f(X)$ that are true (7 with g true and 5 with g false). Thus, the third weighted formula evaluates to 0.25^{12}. There are $12 * 25$ instances of $f(X) \wedge g(Y)$ that are true (12 instances of $f(X)$ paired with 25 instances of $g(Y)$), and thus there are $12 * 25 - 7$ pairs where X and Y are different (subtracting the 7 individuals for which f and g are both true). Similarly, there are $12 * 12$ instances of $f(X) \wedge f(Y)$; each of the 12 individuals can be paired with each of the 12 individuals. The last weighted formula cannot be evaluated because we don't know for how many individuals H is true. Thus, $lrc(Con_1, Fs_1)$ returns:

$$0.1^{18} * 0.25^{12} * 0.3^{12*25-7} * 0.35^{12*12} * lrc(Con_1, Fs_2)$$

where $Fs_2 = \{\langle f(X) \wedge h(X), 0.4\rangle\}$.

The lifted recursive conditioning algorithm, shown in Fig. 6.3, evaluates the factors when they can be evaluated, and branches on counts to make progress to evaluate weighted formulae. Like recursive conditioning (Fig. 5.2), it exploits the graphical structure by recognizing disconnected components and noticing when some values have already been computed.

To branch on a parameterized random variable, $f(X)$, with a single free logical variable, X, on a count $\#_X\phi(X) = n$, means to do case analysis on how many of those n individuals have $f(X)$ true. There are $\binom{n}{i}$ ways that i individuals could be chosen to be true out of the n individuals.

Example 6.7 Consider Con_1 and Fs_1 from Example 6.6. Branching on h for the 7 "X" individuals such that $f(X) \wedge g(X)$ results in

$$lrc(Con_1, Fs_2) = \sum_{i=0}^{7} \binom{7}{i} lrc(Con_2, Fs_2) .$$

where Con_2 is the same as Con_1 but with $\#_X F(X) \wedge G(X) = 7$ replaced with

$$\begin{aligned}&\#_X f(X) \wedge g(X) \wedge h(X) = i,\\ &\#_X f(X) \wedge g(X) \wedge \neg h(X) = 7 - i .\end{aligned}$$

Caching and forgetting work the same as in Fig. 5.2, except that to forget a variable means summing it out of the contexts in which it appears. For example, to forget g from Con_1 gives:

$$\sum_g Con_1 = \{\neg a,\ \#_X f(X) = 12, \#_X \neg f(X) = 18\}$$

```
procedure $lrc(Con :$ context, $Fs :$ set of factors):
    if $\exists v$ such that $\langle\langle Con, Fs\rangle, v\rangle \in cache$
        return $v$
    else if $vars(Con) \not\subseteq vars(Fs)$
        return $lrc(\sum_x Con, Fs)$
    else if $\exists f \in Fs$ such that $f$ can be evaluated in $Con$
        return $eval(f, Con) \times lrc(Con, Fs \setminus \{f\})$
    else if $Fs = Fs_1 \uplus Fs_2$ where $vars(Fs_1) \cap vars(Fs_2) \subseteq vars(Con)$
        return $lrc(Con, Fs_1) \times lrc(Con, Fs_2)$
    else if exists a decomposer for network
        let $Fs'$ be $Fs$ with corresponding variables replaced by a unique constant
        let $n$ be the corresponding population size
        return $lrc(Con, Fs')^n$
    else
        either
            select random variable $x \in vars(Fs) \setminus vars(Con)$
            $sum \leftarrow \sum_{v \in range(x)} lrc(Con \cup \{x = v\}, Fs)$
        or
            select $\#_X\phi(X) = n$ in $Con$ and $f(X)$ not in $\phi(X)$
            $Con' \leftarrow Con \setminus \{\#_X\phi(X) = n\}$
            $sum \leftarrow 0$
            for each $i \in [0, n]$
                $Con'' \leftarrow Con' \cup \{\#_X\phi(X) \wedge f(X) = i, \#_X\phi(X) \wedge \neg f(X) = n - i\}$
                $sum \leftarrow sum + \binom{n}{i} lrc(Con,'' Fs)$
        $cache \leftarrow cache \cup \{\langle\langle Con, Fs\rangle, sum\rangle\}$
        return $sum$
    else (if there is no selection)
        Ground a logical variable in $Fs$ forming $Fs'$
        return $lrc(Con, Fs')$
```

Figure 6.3: Lifted recursive conditioning with an explicit context.

as there are 12 individuals for which f is true, and 18 individuals for which f is true.

To recognize that the grounding is disconnected, there are two cases. If the lifted network is disconnected, the grounding is disconnected. Otherwise, it might be disconnected for each instance.[2] To determine this, we try to find a **decomposer** that decomposes the network into disjoint components. To find this, we replace one of the variables X in a factor with a unique constant c. If c is not in all of the PRVs, then the decomposition fails. If c is in a PRV that unifies with a PRV in another factor, then carry out the unification, and repeat with the new replacement. If this does not fail, we have found a decomposition. All of the instances (with different constants) are disconnected from each other, so we take the model with one constant and raise it to the power of the population size.

To compute the probability of any formula, the formula can form the initial context. To compute the posterior probability of query Q given the observations, we use $P(Q \mid Obs) = P(Q \wedge Obs)/(P(Q \wedge Obs) + P(\neg Q \wedge Obs))$. The calls can share the cache. This means that both Q and Obs are not restricted to be ground, but can include counts. If any population has size n, but has no observations, the initial context should contain $\#_X true = n$. (We have ignored the issue of typing, if there are multiple populations.)

As the population size n of undifferentiated individuals increases.

- If grounding is polynomial—instances must be disconnected—lifted inference is constant in n (taking r^n for real r)
- Otherwise, for unary relations, grounding is exponential and lifted inference is polynomial.
- If non-unary relations become unary, the above holds.
- Otherwise, the algorithm grounds an argument. This is always exponentially better than grounding everything, because it never needs to ground every variable. In the worst case, it grounds all but one of the variables.

We can lift a model that consists just of

$$\langle f(X) \wedge g(Y), \alpha_4 \rangle$$

or just of

$$\langle f(X, Z) \wedge g(Y, Z), \alpha_2 \rangle$$

because the instances of z disconnect the graphs. We can also lift a model of

$$\langle f(X, Z) \wedge g(Y, Z) \wedge h(Y), \alpha_3 \rangle$$

because branching on the number of $h(Y)$ that are true, reduces to the previous case.

[2]Here we ignore the case that arise with binary and above functions where, for example, $f(a, b)$ could be connected to $f(b, a)$ for all a and b but disconnected from other instances; see Poole et al. [2011].

This algorithm cannot completely lift (it is exponential in population size) a model that consists just of:

$$\langle f(X,Z) \wedge g(Y,Z) \wedge h(Y,W), \alpha_3 \rangle$$

or just of:

$$\langle f(X,Z) \wedge g(Y,Z) \wedge h(Y,X), \alpha_3 \rangle \ .$$

The algorithm needs to ground a logical variable, which then results on branching on each member of the population. Note that these simple relational models become complex models in the grounding.

The above lifted recursive conditioning method assumes an undirected model. Directed models also require aggregation. Kisyński and Poole [2009] showed how to perform lifted inference within directed first-order models that require an aggregation operator when a parent random variable is parameterized by logical variables that are not present in a child random variable. Choi et al. [2011a] showed how to perform lifted VE in the presence of aggregate factors such as SUM, AVERAGE, and AND in probabilistic relational models.

Choi et al. [2010] addressed lifted inference when the underlying distributions are Gaussian. Their approach assumes that the model consists of Gaussian potentials. They exploit the fact that the interdependence of Gaussians is essentially pairwise (characterized by the covariance matrix). The same group showed how to realize a **lifted Kalman filter** based on lifted variable elimination (VE) [Choi et al., 2011b]. Lifted inference in continuous models have been applied to market analysis [Choi et al., 2010].

As already mentioned, Gogate and Domingos [2011]'s probabilistic theorem proving reduces the problem of lifted probabilistic inference to weighted model counting in a lifted graph by constructing a set of reduced models, recursively calling itself on each reduced model in this set, and combining the returned values in an appropriate manner. Van den Broeck et al. [2011] employed circuits in first-order deterministic decomposable negation normal form to do the same, also for higher order marginals [Van den Broeck and Davis, 2012]. In related work, Sen et al. [2008] proposed the idea of bisimulated VE grouping together nodes that are indistinguishable in terms of VE computations.

6.3 (LIFTED) APPROXIMATE INFERENCE

In addition to *exact* inference methods (based on variable elimination or search) there are also *approximate* methods and methods for *pre-processing* the data.

From the sampling perspective, there are methods developed based on MCMC algorithm specifically for certain formalisms such as MLNs [Poon et al., 2008] and BLOG [Milch and Russell, 2006]. Milch and Russell [2006] developed an MCMC approach where states are only partial descriptions of possible worlds. Zettlemoyer et al. [2007] extended particle filters to a logical setting. Gogate and Domingos [2011], Venugopal and Gogate [2014] introduced a lifted importance sampling. Niepert [2012] proposed permutation groups and group theoretical algorithms to represent and manipulate symmetries in probabilistic models, which can be used for MCMC.

In one of the first inference methods for relational models, Poole [1993a] explored only part of the search space, returning upper and lower bounds for posterior probabilities where the range between the bounds represents the probability mass of the unexplored space. De Salvo Braz et al. [2009] proposed a method that starts from the query, propagates intervals instead of point estimates, and incrementally grounds the lifted network.

The bisimulation of Sen et al. [2008] has also been extended to approximate inference [Sen et al., 2009]. Essentially, they trade off inference accuracy for computational efficiency, i.e., lifting by e.g., grouping nodes and factors together that are within a user-specified ϵ-distance of each other. Similar ideas have been explored within lifted BP [Kersting et al., 2010, Singla et al., 2010] that we will discuss below.

Recently, Wang et al. [2015] presented ProPPR, in which approximate "local groundings" can be constructed in time independent of database size. Intuitively, ProPPR follows the proof-theoretic approaches biased toward short derivations. More formally, it relates the problem of constructing proofs to computation of personalized PageRank on a linearized version of the proof space, and based on this connection, develops a provably-correct approximate grounding scheme, based on the PageRank-Nibble algorithm.

Last, and somewhat similar in spirit, there are methods for pre-processing [Shavlik and Natarajan, 2009, Mihalkova and Richardson, 2010, Venugopal and Gogate, 2014] that reduce the network size drastically so that ground inference can be performed efficiently on the reduced network.

Let us now illustrate how approximate inference approaches could be lifted using (loopy) belief propagation (BP). Although already quite efficient, BP does not make use of symmetries. Lifted message-passing methods group random variables and factors into sets if they have identical messages.

Example 6.8 Figure 6.4 presents an example of a belief propagation (BP) based lifted inference algorithm. Consider the factor graph with three nodes A, B, and C sending identical messages to the neighbors. As can be seen from the figure, since nodes A and C send and receive the same message, they can be clustered together. B cannot be grouped with A or C since it sends twice the same message and hence is considered to be different from the other two. The resulting compressed graph is shown on the right.

To exploit the symmetries present in the graph structure, **lifted BP** variants [Jaimovich et al., 2007, Singla and Domingos, 2008, Kersting et al., 2009], essentially perform two steps: Given a **factor graph**, they first compute a **lifted factor graph** and then run a modified BP on it. In the first step, lifted BP simulates BP, keeping track of which nodes and factors send the same messages, and groups nodes and factors together correspondingly. Here, initially, all variable nodes and all identical factors fall into corresponding groups as indicated by the colors in Fig. 6.4. Now, each variable node sends its color to its neighboring factor nodes. A factor node collects the

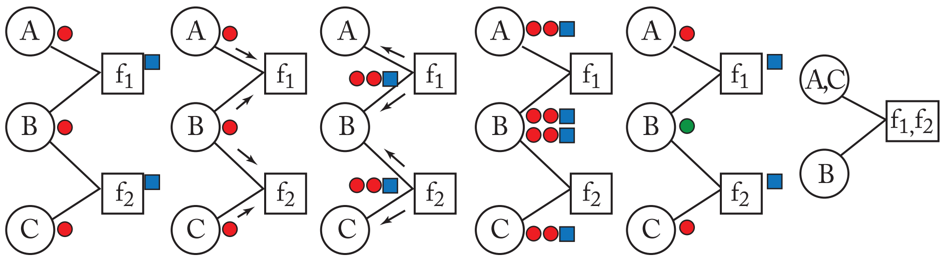


Figure 6.4: Illustration of lifted BP algorithm [Kersting et al., 2009, Ahmadi et al., 2013]. From left to right, the steps of the lifted inference algorithm taken to compress the factor graph assuming no evidence. The shaded/colored small circles and squares denote the groups and signatures produced running the algorithm. On the right-hand side, the resulting compressed factor graph is shown. On this network a modified BP algorithm is employed.

incoming colors, appends its own color at the end, cf. Fig. 6.4, and sends this color signature back to the neighboring variable nodes. The variable nodes stack the incoming signatures together and, hence, form unique signatures of their one-step message history. We group together variable nodes with the same message history and assign new colors to each group. The factors are grouped in a similar way. This **color-passing** process—a.k.a. **color refinement, naive vertex classification**—is iterated until no new colors are created anymore. At convergence, all variables nodes with the same color form a *supernode* and all factors with the same color, a *superfactor*. In the running example, variable nodes `A`, `C` and factor nodes f_1, f_2 are grouped together as shown in Fig. 6.4. This color-passing coincides with the Weisfeiler-Lehman algorithm [Berkholz et al., 2013] and runs in quasi-linear time in the number of nodes and edges in the factor graph. That is, if there are no symmetries to be detected we spend only a small overhead to realize this compared to running BP.

Since supernodes and superfactors are sets of nodes and factors that send and receive the same messages at each step of carrying out BP, we can now simulate BP on the lifted factor graph. An edge in the lifted graph essentially represents multiple edges in the original factor graph. Let $c(\mathfrak{f}, \mathfrak{X}_i)$ be the number of identical messages that would be sent from the factors in the superfactor $\mathfrak{f}$ to each node in the supernode $\mathfrak{X}_i$ if BP was carried out on the original factor graph. The message from a supernode $\mathfrak{X}$ to a superfactor $\mathfrak{f}$ is

$$\mu_{\mathfrak{X}\to\mathfrak{f}}(x) = \mu_{\mathfrak{f}\to\mathfrak{X}}(x)^{c(\mathfrak{f},\mathfrak{X})-1} \cdot \prod\nolimits_{\mathfrak{h}\in \mathrm{nb}(\mathfrak{X})\setminus\{\mathfrak{f}\}} \mu_{\mathfrak{h}\to\mathfrak{X}}(x)^{c(\mathfrak{h},\mathfrak{X})},$$

where $\mathrm{nb}(\mathfrak{X})$ now denotes the neighbor relation of supernode $\mathfrak{X}$ in the lifted graph. The $c(\mathfrak{f}, \mathfrak{X}) - 1$ exponent reflects the fact that a supernode's message to a superfactor excludes the corresponding factor's message to the variable if BP was carried out on the original factor graph. Likewise, the unnormalized belief of any random variable X in $\mathfrak{X}_i$ can be computed as follows $b_i(x_i) = \prod_{\mathfrak{f} \in \mathrm{nb}(\mathfrak{X}_i)} \mu_{\mathfrak{f} \to \mathfrak{X}_i}(x_i)^{c(\mathfrak{f},\mathfrak{X})}$.

However, as the original BP algorithm, lifted BP also does not prescribe a way to solve more complex inference tasks such as computing joint marginals for k-tuples of distant random variables or satisfying assignments of CNFs. A popular solution in these cases is the idea of turning the complex inference task into a sequence of simpler ones by selecting and clamping variables one at a time and running lifted message passing again after each selection. This naive solution, however, recomputes the lifted network in each step from scratch, therefore often canceling the benefits of lifted inference. Online lifting approaches avoid this by reusing already known liftings when computing the lifting of the next inference task [Ahmadi et al., 2010, Nath and Domingos, 2010, Hadiji et al., 2011] and can also be used to realize lifted sampling.

Lifted BP approaches are also appealing because they are simple, efficient, and parallelizable. Moreover, they have paved the way for lifted solutions of many important AI tasks. For instance, one can lift variants of BP for solving satisfiability problems such as survey propagation [Hadiji et al., 2010], which we will discuss later in Chapter 9, or when the underlying distributions are Gaussian [Ahmadi et al., 2011]. Using lifted Gaussian BP, one can realize **lifted Kalman filters**, **lifted PageRank**, **lifted Label Propagation**, and Clustering-on-demand [Ahmadi et al., 2011, Neumann et al., 2011, Hadiji et al., 2015]. Even linear programming solvers can be lifted. Intuitively, given a linear program, we employ a lifted variant of Gaussian BP to solve the systems of linear equations arising when running an interior-point method to solve the linear program. However, this naive solution cannot make use of standard solvers for linear equations and is forced to construct lifted networks in each iteration of the interior-point method again, an operation that can itself be quite costly. Mladenov et al. [2012] showed dow to read off an equivalent linear program of smaller size directly from the original linear program, as we will discuss later in Chapter 9. This lifted linear program can be solved using any off-the-shelf solver.

More importantly, this connects lifted inference and linear program relaxations for the MAP inference problem, see, e.g., Globerson and Jaakkola [2007], and leads to a general, algebraic framework for lifted variational inference [Bui et al., 2013]. The framework was used to lift tree-reweighted BP [Bui et al., 2014], a large class of concave free energies [Mladenov and Kersting, 2015], and several approximate MAP inference algorithms based on message-passing [Bui et al., 2013, Mladenov et al., 2014b,a]. Generally, this line of research together with [Sarkhel et al., 2014] suggests to view lifted inference as standard inference in a reparametrized, (hopefully) smaller model. We take a graphical model, instantiate the evidence, read off the lifted model, and run any probabilistic inference approach we like.

Finally, it should be mentioned that Van den Broeck et al. [2012] built a bridge between lifted exact and approximate inference by lifting the “relax, compensate and then recover” [Choi and Darwiche, 2011].

PART III

Learning

CHAPTER 7
Learning Probabilistic and Logical Models

In order to prepare the stage for learning relational probabilistic models, we first briefly review standard probabilistic and logical learning techniques.

7.1 LEARNING PROBABILISTIC MODELS

When learning probabilistic models, one typically distinguishes two tasks.

- The first task, **parameter estimation**, is concerned with the probabilistic part, i.e., the question **where do the numbers come from?** In this task, it is assumed that the structure (i.e., the graph or logical part) of the representation is fixed and known, and the parameters (i.e., the probabilities or weights that need to be specified as part of the model) have to be estimated.

- In the second task, **structure learning**, the question is **where do the qualitative relationships come from?** In this task, one assumes that neither the structure nor the parameters are fixed but have to be learned.

We also need to consider whether all variables, or only a subset, are **observed**, and whether some data is **missing**. For learning (propositional) probabilistic models, the data is assumed to be a set of tuples, where each tuple has a value for each of the variables. This data is often written as a table with the variables as the columns and the tuples are the rows. Sometimes we will write a tuple as a set of $Variable = value$ pairs.

Example 7.1 To illustrate this, reconsider the burglary alarm example of Fig. 2.1. A possible example for this network could be

$$\begin{gathered}\{burglary = false, earthquake = true, alarm = true\\ johncalls = false, marycalls = true\}\end{gathered}$$

where we have assumed that all variables are observed. This example partially describes a particular situation that arose, namely Judea was called by Mary after an earthquake took place and the alarm sounded.

7.1.1 FULLY OBSERVED DATA AND KNOWN STRUCTURE

For directed models, given the structure, and assuming that all variables are observed, each conditional probability can be learned separately. Learning the conditional probability $P(x \mid y_1, \ldots, y_k)$ where $y_1, \ldots, y_k$ are the parents of x can be treated as a problem of supervised learning where the inputs are the y_i and the output is x. If x has no or few parents (k is small), we can use empirical frequencies, which are the maximum likelihood estimators for the training set. In essence, for conditional probability matrices, the conditional probability $P(x = \hat{x} | y = \hat{x})$ can be estimated as $\frac{count(x=\hat{x}, y=\hat{y})}{count(y=\hat{y})}$. To avoid outcomes with zero probabilities and to avoid overfitting, regularization such as adding Laplace correction, which involves adding pseudo-counts to the data, can be used. When there are more parents, to allow for generalization, any supervised learning technique that can predict probabilities, such as decision trees or neural networks, can be used. Supervised learning can be seen as the task of learning conditional probabilities, and there is a huge literature on this task.

Example 7.2 To illustrate MLE, let us consider tossing a coin. The coin can either be heads or tails. Assume that the evidence is a sequence consisting of n_0 instances of tails and n_1 instances of heads. Suppose $P(\texttt{coin} = \mathit{head}) = p$. Then

$$\log P(E|L, \lambda) \quad = \quad \sum_{e \in E} \log P(e|L, \lambda) = n_1 \cdot \log p + n_0 \cdot \log(1 - p) \ .$$

We can maximize the likelihood or equivalently maximize the log-likelihood by setting the partial derivative of the log-likelihood w.r.t. to p equal to 0.0:

$$\frac{\partial \log P(E|L, \lambda)}{\partial p} = \frac{n_1}{p} - \frac{n_0}{1 - p} = 0 \ .$$

Solving this gives $p = \frac{n_1}{n_0 + n_1}$, which is the empirical proportion of heads which have been seen in the examples. Thus, MLE reduces to frequency counting.

For undirected models, this procedure is conceptually the same where one optimizes the log-likelihood. However, there are major differences to the directed models making it typically harder. Instead of computing counts over individual parameters, we compute the counts over the cliques. Moreover, the factors do not correspond either to probabilities or to conditional probabilities; the only constraint on the parameters in the factor is non-negativity. Another difference is that in the general case of undirected models, the parameters are not decomposed as a product of conditional probability distributions. The normalization constant couples the parameters and effectively rules out a closed-form solution. So, unlike in directed networks, where the parameters can be estimated by simple counting because of local normalization, parameter estimation of undirected models requires non-linear optimization (based, e.g., on gradients) involving inference. If possible, one will construct decomposable models making the optimization of the log-likelihood more efficient.

7.1.2 PARTIALLY OBSERVED DATA WITH KNOWN STRUCTURE

Optimization (involving inference) is generally required—whether directed or undirected—in the presence of missing data. A common method for learning the parameters in this case is the EM algorithm [Dempster et al., 1977, McLachlan and Krishnan, 1997]. The EM algorithm intuitively works by filling in the missing information using the model. EM starts with a model, or with a random completion of data and iteratively perform the following two steps until convergence.

E-Step: Using the current parameters of the model, complete each tuple by computing a distribution over the unobserved variables conditioned on the observed variables for that tuple. This step requires inference.

M-Step: Using each completion as a fully observed data case, weighted by its probability, compute the updated parameter values using (weighted) frequency counting.

The frequencies over the completions are called the *expected counts*.

To illustrate EM, consider a Naive Bayes example where y is the parent (in Bayes net terminology) of variables $x_1, \ldots, x_n$ where we observe all the x values but y's values are not observed. In the E-step, the EM algorithm would compute for each training example k the distribution of y:

$$P(y(k) = m|x_1(k), \ldots, x_n(k)) = \frac{P(y(k) = m)\Pi_i P(x_i(k)|y(k) = m)}{\sum_j P(y(k) = j)\Pi_i P(x_i(k)|y(k) = j)}$$

using the current parameter settings of all the conditional probability tables. Now using these pseudo counts, the algorithm applies MLE to the parameters with one change: instead of using the actual counts as is done in MLE, we use the expected counts from the previous step. Hence, in the M-step, we obtain

$$P(x_i = j|y = m) = \frac{\sum_k P(y(k) = m|x_1(k), \ldots, x_n(k)) \times \mathcal{I}(x_i(k) = j)}{\sum_k [P(y(k) = m|x_1(k), \ldots, x_n(k))]},$$

where $\mathcal{I}(x_i(k) = j)$ is 1 if $x_i(k) = j$ is observed and is 0 otherwise. The E and M steps are iterated until the probabilities converge.

EM can also be used for arbitrary missing data, but relies on the missing at random assumption, which is rarely appropriate. It is almost always better to model why the data is missing [Marlin et al., 2011, Mohan and Pearl, 2014].

7.1.3 UNKNOWN STRUCTURE AND PARAMETERS

For structure learning in graphical models, it is standard to employ a greedy approach. Initially, we start with a candidate structure (from either prior knowledge or a prior structure). We then score this candidate structure, and consider all possible local changes to the structure by adding to, deleting from or reversing edges in the graphical model, and take the best refinement. If the refined model has a better score than the current candidate structure, we continue the process from the refined model until we reach a local optimum.

Hence, the next question is how do we score the candidate structures? As score, we typically optimize model simplicity plus fit to data. The idea is that since the training data may not be complete, simply relying on the data can lead to a very dense model that overfits the data. So most algorithms use a term that penalizes complex structures, and try to minimize

$$P(\mathit{model} \mid \mathit{data}) \propto P(\mathit{data} \mid \mathit{model}) * P(\mathit{model})$$

or equivalently minimize the log of the probability:

$$\log P(\mathit{model} \mid \mathit{data}) \propto \log P(\mathit{data} \mid \mathit{model}) + \log P(\mathit{model}) \ .$$

Determining the probability of the data given the model requires inference. The log of the probability of the model can be estimated by the number of bits required to represent the model. Note that model consists of the structure and the parameters and the score is computed after fitting the parameters using MLE for the completely observed case or EM for the partially observed case. Once the score is computed, the best refinement is performed. And the next refinement is searched for until the score no longer improves. An example of this is presented in Fig. 7.1. As can be observed, there are three nodes with an initial configuration. There are potentially three different operations considered, one of which yields a higher score than the current one. The search then proceeds down that path to make more local changes to the underlying graph.

One common approximation is to estimate $\log P(\mathit{model})$ in terms of the number of real-valued parameters r. If there are n data points, there are n different probabilities to distinguish, and so each parameter can be represented using $\log n$ bits. Thus, an estimate of the $\log P(\mathit{model})$ is $r \log n$. This approximation is related to the Bayesian information criterion (BIC) [Schwarz, 1978], which is justified by more sophisticated arguments than presented here, resulting in using $0.5 r \log n$.

More information on learning probabilistic graphical models can be found in Koller and Friedman [2009]. We now turn our attention to learning logical models.

7.2 LOGICAL AND RELATIONAL LEARNING

Learning logical formulae is the topic of purely (i.e., non-statistical) relational learning and of the field of **inductive logic programming** (**ILP**) [Muggleton and De Raedt, 1994, De Raedt, 2008]. Various settings exist depending on the class of logical formulae being learned as well as the way the data is represented. However, the general learning problem is cast as that of finding a hypothesis H (a logical formula) from a set of **positive** and **negative examples** *Pos* and *Neg* such that H *covers* all positive and no negative examples. This is a classical **concept-learning** task where the goal is to obtain a description of the unknown target concept that classifies all the examples correctly [Mitchell, 1997]. The learned hypotheses can then be used to predict the class of the examples and evaluated using typical measures such as accuracy on unseen data.

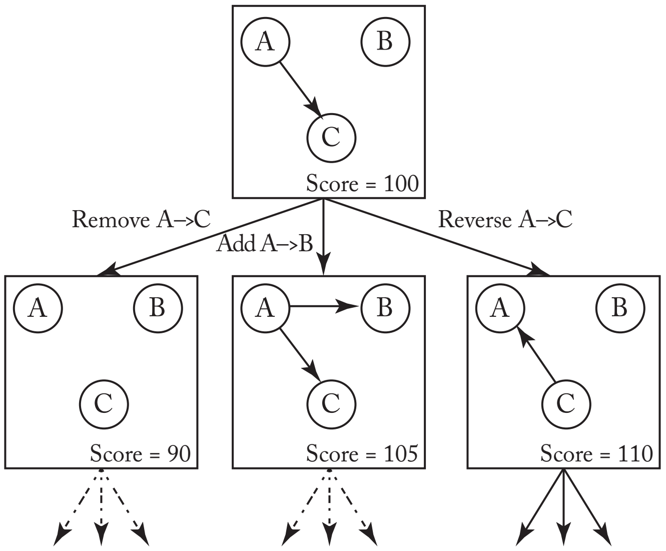


Figure 7.1: Example of how structure learning proceeds as local search. Initially, the model has an arc from A to C and then local refinements are performed to compute the score of each local refinement. Then the highest scoring one is chosen and the search continues after making that refinement.

Two ILP settings, that we will now introduce, are specifically relevant for SRL: learning from interpretations and learning from entailment. They make different choices for what concerns the nature of the examples and the coverage relation. Their difference is also relevant to different learning procedures typically used in our primary probabilistic logic models, MLNs and Problog.

We will first introduce the settings in propositional logic (following De Raedt [2010]) and then briefly sketch what changes when working in first order logic.

7.2.1 TWO LEARNING SETTINGS

When **learning from interpretations**, hypotheses are logical formula, typically sets of clauses, and examples are interpretations. For propositional theories, interpretations are assignments of truth-values to propositional variables. We specify interpretations through the set of propositional variables that are true. An interpretation is a model for a hypothesis if it satisfies all clauses in the hypothesis. When learning from interpretations, a hypothesis h **covers** an interpretation if and only if the interpretation is a model for the hypothesis.

Example 7.3 Consider the following interpretations:

$$\{blackbird, bird, normal, flies\}$$
$$\{ostrich, small\}$$

and the hypothesis

$$\begin{aligned} \mathit{flies} &:- \mathit{bird}, \mathit{normal}.\\ \mathit{bird} &:- \mathit{blackbird}.\\ \mathit{bird} &:- \mathit{ostrich}. \end{aligned}$$

The first interpretation is a model for the hypothesis but the second is not. Because the condition part of the rule *bird* :– *ostrich* is satisfied in the second interpretation (as it contains *ostrich*), the conclusion part, that is *bird*, should also belong to the interpretation in order to have a model.

In **learning from entailment**, both hypotheses and examples are logical formulae, typically *definite clauses*. When learning from entailment, a hypothesis h **covers** an example e if and only if $h \models e$; that is when h logically entails e, or equivalently, when e is a logical consequence of h.

Example 7.4 Reconsider the hypothesis from the previous example and the following two examples:

$$\begin{aligned} \mathit{flies} &:- \mathit{blackbird}, \mathit{normal}, \mathit{small}.\\ \mathit{flies} &:- \mathit{ostrich}, \mathit{small}. \end{aligned}$$

The first of these clauses is covered by the hypothesis because it is a logical consequence of the hypothesis. On the other hand, the second example is not covered.

The goal in both settings is to induce a formula that covers all positive and none of the negative examples; when learning from intepretations this is defined in terms of the interpretations that satisfy the hyopthesis, when learning from entailment in terms of clauses that should be entailed by the hypothesis. Both settings have been used in ILP and there is a subtle difference in meaning between the two representations. By representing the birds using an interpretation, it is assumed that all propositions not in the interpretation are false. Thus, in Example 7.3, the interpretations imply, for instance, that the proposition *insect* is known to be false. This assumption is not made using the clausal representation of the bird. A further difference is that in the clausal representation, there is typically a distinguished predicate, the predicate *flies*, that is entailed by the set of conditions. In contrast, using interpretations, all predicates are treated uniformly. The former representation can be more natural when learning a specific concept as a predicate definition, such as the concept of flying things; the latter representation is more natural to describe a set of characteristics of the examples, such as the baskets bought in a supermarket.

7.2.2 THE SEARCH SPACE

The typical learning problem tackled in logical learning is that of finding a hypothesis that covers all positive and none of the negative example. As already argued above, this is essentially a concept-learning problem [Mitchell, 1997] and so the same principles and algorithms apply. In particular, concept and rule-learning algorithms exploit the *generality order* on the search space. A hypothesis g is **more general than** a hypothesis s whenever all examples covered by s are also covered by g.

One of the key properties exploited in ILP is that, when using logic, the generality relation coincides with that of logical entailment. But it also interesting to observe that the learning setting (entailment or interpretations) determines the direction of the generality relation. Indeed, when learning from entailment, g is more general than s if and only if $g \models s$, while when learning from interpretations, g is more general than s if and only if $s \models g$.

Example 7.5 Consider using the following two clauses h_1 and h_2 as the hypotheses:

$$\begin{array}{ll} h_1 : & \mathit{flies} \ \text{:--}\ \mathit{blackbird}. \\ h_2 : & \mathit{flies} \ \text{:--}\ \mathit{blackbird}, \mathit{normal}. \end{array}$$

Treat each clause as a set of (implicitly disjointed) literals:

$$\begin{array}{lll} h_1 & = & \{\mathit{flies}, \neg \mathit{blackbird}\} \\ h_2 & = & \{\mathit{flies}, \neg \mathit{blackbird}, \neg \mathit{normal}\} \ . \end{array}$$

The first clause logically entails the second one; that is, $h_1 \models h_2$. In propositional logic, it is easy to see that $h_1 \models h_2$ as $h_1 \subseteq h_2$ when views as sets. We say that h_1 **subsumes** h_2.

When learning from entailment, clause h_1 generalizes h_2. The reason is that any example clause e covered by h_2 must satisfy $h_2 \subseteq e$, and whenever this holds, also $h_1 \subseteq e$ holds, hence, h_1 covers e. On the other hand, when learning from interpretations, h_2 is a generalization of h_1 as $h_1 \models h_2$ and any model of h_1 is a model of h_2.

The entailment or coverage relation imposes a structure on the search space, which is formalized in the notion of subusmption. The propositional subsumption relation induces a complete lattice on the space of possible clauses. A *complete lattice* is a partial order—a reflexive, anti-symmetric and transitive relation —where every two elements posses a unique least upper and greatest lower bound. An example lattice for rules defining the predicate *flies* in terms of *bird*, *normal* and *small*. is illustrated in the Hasse diagram depicted in Fig. 7.2.

The Hasse diagram also visualizes the different *refinement* operators that can be used. The *downward* operator ρ_d maps a clause to the set of its children in the diagram, i.e., it adds a single literal, whereas the *upward* operator ρ_u maps a clause to the set of its parents.

In addition to using stepwise refinements, some systems like Golem [Muggleton and Feng, 1990] also exploit the properties of the underlying lattice by computing the least upper bound of two formulae. The least upper bound operator is known under the name of *least general generalization* (*lgg*) in the machine learning literature. Using set notation for clauses $lgg(c_1, c_2) = c_1 \cap c_2$. The *lgg* is the least common ancestor in the Hasse diagram.

7.2.3 TWO ALGORITHMS: CLAUSAL DISCOVERY AND FOIL

We now sketch two algorithms for inducing logical theories from examples, one for each setting.

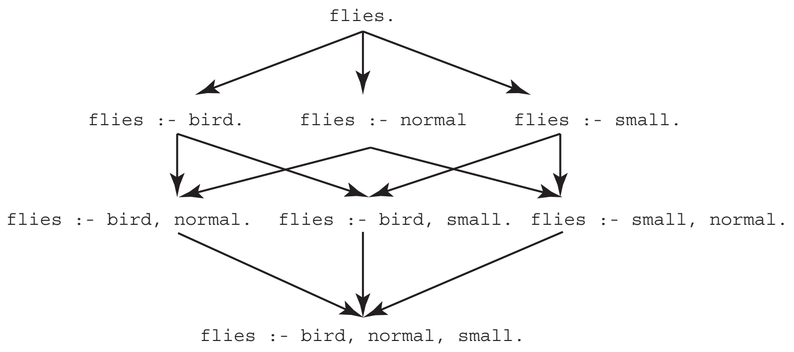


Figure 7.2: The Hasse diagram for the predicate `flies` (from De Raedt [2010]).

Clausal discovery: learning from interpretations. The clausal discovery algorithm, based on the work by Valiant [1984] and De Raedt and Dehaspe [1997], learns from interpretations and uses only positive examples. It computes the maximally specific set of clauses that covers (i.e., is satisfied by) all of the given positive examples. The clausal discovery setting can be viewed as that of learning a set of constraints. This is reminiscent of Markov Logic where formulae act as soft constraints. The only difference is that in clausal discovery hard constraints are learned, which correspond to a deterministic Markov Logic theory (where all weights would be set to infinity). We first illustrate the problem setting and then sketch an algorithm.

Example 7.6 (Clausal discovery) Consider the following three positive examples specified as interpretations:

$$\begin{array}{l} \{male, human\} \\ \{female, human\} \\ \{\} \end{array}$$

The clausal discovery algorithm would then construct the following set of clauses (assuming the only predicates are those occurring in the interpretations):

$$\begin{array}{rl} human & :- \; male. \\ human & :- \; female. \\ false & :- \; male, female. \\ female; male & :- \; human. \end{array}$$

Analyzing these clauses in terms of the Hasse diagram reveals that each clause contains a minimal set of literals so that all examples are covered. Furthermore, it contains all such clauses.

```
procedure $CD(D$ : set of interpretations, $A$ : set of propositions):
    $H := \{\}$
    $Q := \{\emptyset\}$
    while $Q$ not empty do
        delete clause $c$ from $Q$
        if $c$ covers all examples in $D$ and no subset of $c$ covers $D$
        then add $c$ to $H$
        else add all downward refinements $d$ of $c$ using $A$ for which $d \preceq_{LEX} c$ to $Q$
    return $H$
```

Figure 7.3: A clausal discovery algorithm.

This set of clauses can be found by Algorithm 7.3, which initializes its queue of candidates with the empty clause; that is, it starts at the top of the lattice. It then repeatedly deletes a clause, checks whether it covers the examples and is minimal, and if so, adds the clause to the hypothesis. If it does not, it adds the downward refinements of the clause to the queue, which are obtained by adding single literals. In order to avoid generating the same clause more than once, a lexicographic order can be imposed on the set of literals.

Learning from entailment   Learning from entailment is by far the most popular setting in ILP. Systems typically learn a set of rules for a single predicate from positive as well as negative examples. They are based on rule-learning principles and essentially employ a separate-and-conquer approach. For instance, FOIL [Quinlan, 1990], starts from the empty hypothesis, and then repeatedly searches for one rule that covers as many positive examples as possible and no negative example, adds it to the hypothesis, removes the positives covered by the rule, and then iterates. This process is continued until all positives are covered. To find one rule, it performs a hill-climbing search through the space of clauses ordered according to generality. The search starts at the most general rule, the one stating that all examples are positive (for the *flies* example this would be the top clause in the latice), and then repeatedly specializes it. Among the specializations it then selects the best one according to a heuristic evaluation based on information gain. A heuristic, based on the minimum description length principle, is then used to decide when to stop specializing clauses. Numerous variations on rule learning algorithms exist, depending on the search strategy (e.g., hill-climbing, beam-search, bottom-up, or top-down), and the heuristics used.

7.2.4 FROM PROPOSITIONAL TO FIRST-ORDER LOGIC

When working with relational or first-order logic rather with propositional logic, the key change is that the structure on the search space becomes more complicated as one has to deal with variables, functors, constants, and substitutions. The standard way of dealing with this is to use the θ-

subsumption framework of Plotkin [1970]. It provides a generalization relation for clausal logic and it extends propositional subsumption to first-order logic.

While downward refinements with propositional subsumption are obtained by adding a literal to a clause, with θ-subsumption one can also apply a substitution. More formally, for two clauses g and s, g θ-subsumes s if and only if $\exists$ a substitution $\theta : g\theta \subseteq s$. For instance, the first clause for *likes* subsumes the second one with the substitution $\{Y/tom\}$. Consider the following three positive examples specified as interpretations:

$$\begin{aligned} likes(X, Y) &\ :- \ neighbors(X, Y). \\ likes(X, tom) &\ :- \ neighbors(X, tom), male(X). \end{aligned}$$

By applying downward refinements, one can obtain the second clause by first adding a literal $male(X)$ and then applying the substitution.

θ-subsumption has some interesting properties. First, θ-subsumption is sound; that is, g θ-subsumes s implies that $g \models s$. The other direction does not hold for self-recursive clauses involving structured terms (e.g., $nat(s(X)) \ :- \ nat(X)$). Second, deciding θ-subsumption is an NP-complete problem. Third, θ-subsumption is reflexive, transitive but unfortunately not anti-symmetric, which can be seen by consider the clauses

$$\begin{aligned} parent(X, Y) &\ :- father(X, Y). \\ parent(X, Y) &\ :- father(X, Y), father(U, V). \end{aligned}$$

The first clause clearly subsumes the second one because it is a subset. The second one subsumes the first with the substitution $\{X/U, V/Y\}$. The two clauses are therefore equivalent under θ-subsumption, and hence also logically equivalent. The loss of anti-symmetry complicates the search process. The naive application of a downward specialization may yield syntactic specializations that are logically equivalent. This is illustrated above where the second clause for *parent* is a refinement of the first one obtained by adding a literal. In this way, useless clauses are generated and there is a danger that if the resulting clauses are further refined, the search will end up in an infinite loop. Plotkin [1970] studied the quotient set induced by θ-subsumption and proven that it is a complete lattice; see also De Raedt [2008, 2010] for more details.

7.2.5 AN ILP EXAMPLE

As final information about inductive logic programming, we provide an example of the use of FOIL inspired by the mutagenicity dataset [Srinivasan et al., 1996], which is a well-known application of ILP.

Example 7.7 (FOIL) Rather than working with clauses as examples, FOIL requires that examples are true and false ground facts, and assumes that further information is given in the background theory, which FOIL restricts to a set of ground facts. The goal then is to induce a hypothesis (a set of clauses) that covers all positive and none of the negative examples.

Assume the following facts (describing part of molecule 225) are contained in the background theory. Let us call this set of facts B.

$$
\begin{array}{ll}
\mathit{molecule}(225). & \mathit{bond}(225, \mathit{fl\_1}, \mathit{fl\_2}, 7). \\
\mathit{logmutag}(225, 0.64). & \mathit{bond}(225, \mathit{fl\_2}, \mathit{fl\_3}, 7). \\
\mathit{lumo}(225, -1.785). & \mathit{bond}(225, \mathit{fl\_3}, \mathit{fl\_4}, 7). \\
\mathit{logp}(225, 1.01). & \mathit{bond}(225, \mathit{fl\_4}, \mathit{fl\_5}, 7). \\
\mathit{nitro}(225, [\mathit{fl\_4}, \mathit{fl\_8}, \mathit{fl\_10}, \mathit{fl\_9}]). & \mathit{bond}(225, \mathit{fl\_5}, \mathit{fl\_1}, 7). \\
\mathit{atom}(225, \mathit{fl\_1}, c, 21, 0.187). & \mathit{bond}(225, \mathit{fl\_8}, \mathit{fl\_9}, 2). \\
\mathit{atom}(225, \mathit{fl\_2}, c, 21, -0.143). & \mathit{bond}(225, \mathit{fl\_8}, \mathit{fl\_10}, 2). \\
\mathit{atom}(225, \mathit{fl\_3}, c, 21, -0.143). & \mathit{bond}(225, \mathit{fl\_1}, \mathit{fl\_11}, 1). \\
\mathit{atom}(225, \mathit{fl\_4}, c, 21, -0.013). & \mathit{bond}(225, \mathit{fl\_11}, \mathit{fl\_12}, 2). \\
\mathit{atom}(225, \mathit{fl\_5}, o, 52, -0.043). & \mathit{bond}(225, \mathit{fl\_11}, \mathit{fl\_13}, 1). \\
\ldots & \\
\mathit{ring\_size\_5}(225, [\mathit{fl\_5}, \mathit{fl\_1}, \mathit{fl\_2}, \mathit{fl\_3}, \mathit{fl\_4}]). & \\
\mathit{hetero\_aromatic\_5\_ring}(225, [\mathit{fl\_5}, \mathit{fl\_1}, \mathit{fl\_2}, \mathit{fl\_3}, \mathit{fl\_4}]). & \\
\ldots &
\end{array}
$$

Consider the positive example $e = \mathit{mutagenic}(225)$. Running the hill-climbing search through the space of clauses ordered according generality described above, we may find that this example is covered by the clause

$$\mathit{mutagenic}(M) \quad :- \quad \mathit{nitro}(M, R1), \mathit{logp}(M, C), C > 1$$

as the above clause subsumes the clause $e \ :- B$ that represents the example. In the ILP literature, one would rather write that $H \cup B \models e$. To see this, we unify $\mathit{mutagenic}(225)$ with the clause's head. This yields

$$\mathit{mutagenic}(225) \quad :- \quad \mathit{nitro}(225, R1), \mathit{logp}(225, C), C > 1 \ .$$

Now, $\mathit{nitro}(225, R1)$ unifies with the fifth ground atom (left-hand side column) in B, and $\mathit{logp}(225, C)$ with the fourth one. Because $1.01 > 1$, we found a proof of $\mathit{mutagenic}(225)$. So, H would be a valid hypothesis if the dataset contained only e.

In practice, many more examples would be given which the learned hypothesis should discriminate. Assuming that the above clause does not cover any negative example, we remove all positive examples covered by the clause (such as $\mathit{mutagenic}(225)$) and learn the next rule on the remaining examples, and so on.

CHAPTER 8

Learning Probabilistic Relational Models

So far, we have assumed that we were given a relational probabilistic model, i.e., both logical structure and the parameters were assumed to be given. Usually, however, this assumption does not hold and we have to construct the model. Doing this by hand is usually problematic as this requires too much domain knowledge and it is often also unclear which values for the parameters to chose. A more practical approach is to learn the model from data. That is, if we have access to a set of examples, we can learn the model parameters for a fixed structure, or learn (some or) all of the logical structure of the model in addition to the parameters.

This chapter briefly introduces the key issues and approaches for learning probabilistic relational models.

8.1 LEARNING AS INFERENCE

If we have a relational probability model and condition on observations about some of the individuals, we can learn about other individuals and about other properties of the observed individuals. This learning is just a form of inference.

Example 8.1 Consider the model of Fig. 3.1, with appropriate probabilities, and conditioning on the database of Fig. 1.2. In this case, we could learn that student S_1 is intelligent and so course C_2 is difficult. Similarly we could learn that student S_2 is not intelligent, and so course C_3 is not difficult. These, in turn, provide information about the intelligence of students S_3 and S_4 as the model conditioned on these observations would make different predictions about how S_3 and S_4 would do in course C_4, even though they have exactly the same averages, the courses they took have the same averages, and we have no observation on course C_4.

This problem has been studied in terms of reference classes [Reichenbach, 1949], where the use of the narrowest reference class for which there are adequate statistics has been advocated [Kyburg, 1983]. However, a simple variant of the above example shows why a narrow interpretation of this does not work:

Example 8.2 Suppose we want to predict how Sam would do on a course she hasn't taken, say CS333. Here the narrowest reference class might mean to use the grade of Sam, the average grade in the course CS333 or some mix of all reference classes. Suppose that the average grade of all students in all courses was 65, and that the average grade of Sam was 80 and the average grade of CS333 was also 80. If we are to use only the range of statistics available, then we might be tempted to predict somewhere between 65 and 80 as the grade of Sam. However, it is likely that Sam will get a higher grade than 80; after all Sam is a well-above-average student and CS333 is an easy course.[1] Such reasoning requires going beyond the narrowest class of reference as we show in this chapter.

Example 8.3 Inferring the posterior distribution of the θ parameters of Fig. 3.3 conditioned on the observations is a way of learning the parameters for a relational probabilistic model [Buntine, 1994, Jordan, 2010]. Unfortunately, inference for such models is very difficult.

8.2 THE LEARNING PROBLEM

A more common view on learning denotes the process by which one adapts (probabilistic, relational) models on the basis of data. When learning probabilistic relational models (as with the more traditional probabilistic representations presented in the previous chapter), one typically distinguishes parameter estimation from structure learning.

It will turn out that parameter estimation techniques for probabilistic relational models are based on the same principles as those of learning propositional graphical models (cf. Section 7.1), the key differences being that we need to tie the parameters together and take into account aggregation and/or combination rules (cf. Section 4.7). Furthermore, structure learning techniques have to search a space of parametrized logical formulae and are therefore very much related to inductive logic programming (cf. Section 7.2). Because one also needs to estimate the parameters when learning the structure of a probabilistic model, we will introduce parameter estimation before structure learning. But before doing so, we will first clearly define the data for relational models. Here, it will turn out to be useful to distinguish different learning settings as in inductive logic programming based on the type of data that is available. This will be akin the differences between **learning from interpretations** for relational graphical models and **learning from entailment** for probabilistic programs.

8.2.1 THE DATA USED

Different forms of data are used for learning. Which form of data to use depends on the relational model and the application.

Data for Probabilistic Relational Models

An example or data case used for learning in a Bayesian network setting consists of the random variables specified in the network and a possibly partial assignment of values to the variables, cf. Example 7.1.

[1]It could also be the case that the average for CS333 is high because CS333 is only taken by the top students. This issue is discussed by Pearl [2009, Ch.6] in reference to Simpson's paradox.

Student	*Course*	*Grade*
s_1	c_1	*a*
s_2	c_1	*c*
s_1	c_2	*b*
s_2	c_3	*b*
s_3	c_2	*b*
s_4	c_3	*b*
...	...	...

Course	*Difficulty*
c_1	*high*
c_2	*low*
c_3	*high*
...	...

Student	*Intelligence*
s_1	*high*
s_2	*low*
...	...

Figure 8.1: The dataset of Fig. 1.2 with observed properties.

For relational probabilistic models, such as PRMs [Getoor, 2001], BLPs [Kersting and De Raedt, 2007], and Markov Logic, it is natural to work with examples in the form of (partial) interpretations, which directly upgrades the Bayesian network setting illustrated in Example 7.1. Indeed, the atoms in the interpretations specify the random variables, and their values specify the state they are in. An example of this data would be a partial data base (as shown in Fig. 8.1). This is reminiscent of the *learning from interpretations* setting of inductive logic programming.

Data for Probabilistic Programs

When considering probabilistic (logic) programs such as alarm and smokers used in Section 3.2.2 to introduce ProbLog, it is natural to focus on examples for a single predicate (such as alarm or smokers) and to use the *learning from entailment* setting from inductive logic programming. In this view, the examples correspond to facts such as *smokes*(*sam*) and *calls*(*jeff*). If the examples are positive, they should be (probabilistically) entailed by the target program, if they are negative, they should not be entailed (or otherwise have a very low probability). This setting has been used in systems such as PRISM [Sato, 1995] and ProbLog [Fierens et al., 2015].

Learning from Interpretations or from Entailment

As already indicated in Section 7.2, learning from entailment and from interpretations are two different settings. Learning from entailment fits in with the proof-theoretical view of logic, while learning from intepretations with a model-theoretic one. The key difference between the two settings is that in a (partial) interpretation all relations are equal as there is no distinguished target predicate, while when learning from entailment (from facts or clauses), there typically is a distinguished target predicate (corresponding to the head of the clause). When the target predicate or fact is known, a partial interpretation I can be transformed into a clause $target :- I - \{target\}$ and one can learn from entailment.

In a graphical model context, one will almost always use partial interpretations, while in inductive logic programming, the most popular setting is that of learning from entailment, which partially explains why initially the logic programming based systems used the learning entailment setting.

8.3 PARAMETER LEARNING OF RELATIONAL MODELS

As for the propositional case, different techniques are used according to whether the data is fully observable or not. Parameter learning methods assume that the underlying logical structure of the model is fully known. The parameters (weights/probabilities) have to be estimated from the data.

8.3.1 FULLY OBSERVABLE DATA

When all of the relations for a model are observed, we can estimate the parameters or conditional probabilities by counting and/or by adapting supervised learning techniques [Sato and Kameya, 1997, Friedman et al., 1999, Kersting and Driessens, 2008, Natarajan et al., 2009, Jaeger, 2007]. Essentially these techniques correspond to computing the maximum likelihood estimates (MLE) subject to the parameter tying constraints—where instances of the same variable inside the same clause share the parameters.

Example 8.4 Consider learning the parameters of a model such as in Fig. 3.1 from a dataset such as in Fig. 8.1, where all of the relations have been observed.

One issue that arises in relational domains is that we need to be clear about the meaning of the probability of a grade in a course. There seems to be two alternate readings.

- It means the probability that the student took the course and got the particular grade. In this case, we need an extra value in the range of the grade variable representing the "did not take" state. Now we can estimate, for example, the probability $P(gr(S, C) = a \mid i(S) = high, d(C) = high)$ as the proportion of a's in the data:

$$\frac{\left|\{\langle s, c\rangle : gr(s, c) = a, i(s) = high, d(c) = high\}\right|}{\left|\{\langle s, c\rangle : i(s) = high, d(c) = high\}\right|},$$

  where the sizes can include pseudo-counts to represent prior information, e.g., to carry out Laplace smoothing [Gelman et al., 2004, p.36]. This formula will give a low probability for each grade for the case where students only take a small proportion of the cases.

- It means the probability the student would get a grade if they took the course, i.e., what is their expected grade if the student took the course. In this case, to estimate probabilities we only use those pairs that have recorded grades. For example, we can estimate the probability $P(gr(S, C) = a \mid i(S) = high, d(C) = high)$ as:

$$\frac{\left|\{\langle s, c\rangle : gr(s, c) = a, i(s) = high, d(c) = high\}\right|}{\left|\{\langle s, c\rangle : i(s) = high, d(c) = high, \exists G\, gr(s, c) = G\}\right|}.$$

This estimate suffers from the missing data problem; it is unlikely that the data is missing at random.

The example indicates also the difference between propositional and relational probabilistic models. If there is only a single course and student, there will only be a single $< s, c >$ pair, then the plates model corresponds to a propositional Bayesian network, and all the parameters provide information about this single student s and course c. If there are more courses $c_1, \ldots, c_k$ and students $s_1, \ldots, s_n$, then the (random) variables $i(s_i)$, respectively, $d(c_j)$ and $g(s_i, c_j)$ will share the same parameters (to exploit the symmetries between the students and between the courses), and so the counts will have to sum also over the students s_i and courses c_j. It is easy to generalize the MLE derivation of Example 7.2 to formally show the that the empirical frequencies are the maximum likelihood estimates.

Another possibility is to use supervised learning techniques (such as decision trees) to learn richer representations of how a variable depends on its parents, cf. also Section 8.4.3.

8.3.2 PARTIALLY OBSERVED DATA

As mentioned in the previous chapter, in the presence of missing data, the maximum likelihood estimate typically cannot be written in closed form. It is a numerical optimization problem, requiring nonlinear optimization. The most commonly adapted technique, as with the propositional case, for learning probabilistic relational models is the **Expectation-Maximization** (**EM**) algorithm [Dempster et al., 1977, McLachlan and Krishnan, 1997] discussed in Section 7.1 although gradient-based approaches have also been used, cf. [Domingos and Lowd, 2009, Kersting and De Raedt, 2001, 2007, Natarajan et al., 2009].

Let us now discuss how EM has been applied within the various relational probabilistic models. For the sake of simplicity, we will only state the key ideas and will not address any advanced issues such as efficiency concerns. We first discuss the knowledge-based model construction approaches, which extend graphical models with relations, and then probabilistic programs.

Relational Graphical Models

Parameter estimation for probabilistic rules [Koller and Pfeffer, 1997], probabilistic relational models [Friedman et al., 1999, Getoor, 2001, Getoor et al., 2001a], and Bayesian logic programs [Kersting and De Raedt, 2001, 2007], as well as Markov logic networks [Domingos and Lowd, 2009] all follow the same principle: the given data and the current model induce via grounding a graphical model explaining each data case. Then, the parameters of the induced graphical model are estimated using standard parameter estimation methods, such as those discussed in Heckerman [1995], Koller and Friedman [2009] and the previous chapter. All nodes of the graphical model, which do not occur in the evidence but are introduced by the program, are not observed, i.e., they are assigned a question mark. Thus the evidence corresponds to a partial joint state of the resulting graphical model. It is completed by the E-step using standard inference procedures (cf. Chapter 6) and it can be used to determine the distribution of the values for the unobserved states.

To continue our illustration about grading, consider that we know $gr(an, algorithms) = a$, $d(algorithms) = high$ but do not know the values of $i(an)$ and $gr(an, ai)$. We can then compute the distribution of

$$P(i(an), gr(an, ai) | gr(an, algorithms) = a, d(algorithms) = high)$$

and complete the examples using the current estimates in the conditional probability distributions of the model. The resulting values can then be used as weights on the completed examples and used for counting. Once the counts are obtained, the M-step can compute the improved estimates of the parameters in the conditional probability distributions. The difference with standard parameter estimation is that parameters for different nodes in the network—those corresponding to different ground instances of the same template—are tied together, that is, forced to be identical. This technique is akin to that in recurrent neural networks [Williams and Zipser, 1995] or dynamic Bayesian networks [Dean and Kanazawa, 1988]. However, it requires a unique assignment of parent nodes to templates.

For directed relational models such as PRMs, aggregation functions guarantee this unique assignment requirement as the aggregated examples constitute the states of the nodes in the induced Bayesian network. Note that PRMs are inspired from the data base literature and hence they employ the notion of aggregators. On the other hand, in case of the probabilistic logic models like PLPs and BLPs, uniqueness is enforced by using decomposable combining rules [Koller and Pfeffer, 1997, Kersting and De Raedt, 2001, Natarajan et al., 2009]. The effect of a decomposable combining rule can be represented using extra nodes in the induced Bayesian network. Most combining rules commonly employed in Bayesian networks such as *noisy or*, *noisy and*, mean, weighted mean or linear regression are decomposable. Jaeger [2007] and Natarajan et al. [2009] adapted and refined the EM approaches to different representations and combining rules. Natarajan et al. [2009], in particular, presented the different update equations in the E and M steps when using weighted mean, mean and noisy or combining rules.

Recently, Ahmadi et al. [2013] employed lifted belief propagation (covered in Chapter 6) for parameter estimation. This is tricky as the symmetries within relational models can easily break since variables become correlated by virtue of depending asymmetrically on evidence. An appealing idea for such situations is to train and recombine local models. This breaks long-range dependencies and allows to exploit lifting within and across the local training tasks. Moreover, as Ahmadi et al. [2013] showed, it naturally paves the way for the scalable lifted training approaches based on stochastic gradients, both in an online and a MapReduced fashion. Van Haaren et al. [2015] showed how to exploit exact lifted inference when (structure) learning MLNs. Wang et al. [2015] presented a fast and easily parallelized weight-learning algorithm for ProPPR based on random walks.

Probabilistic Programs

The above approaches were knowledge-based model construction approaches, which use the logic as a template to induce a ground graphical model, a Bayesian, or Markov network, which can then be used for defining the semantics for inference and for learning. On the other hand, when considering probabilistic logic programs in ICL, PRISM, or ProbLog, the semantics is defined more in logical terms, that is, in terms of model and proof theory, which can also be used as the basis for inference. We have seen that there are at least two ways to exploit this. The first was based on a reduction to weighted model counting (cf. Section 6.1.3), the second on the use of probabilistic abduction (Section 3.2.2). In both cases, one obtained a logical formula f which is then typically compiled into a logical form or circuit (such as an OBDD, a d-DNNF, or an SDD) [Darwiche and Marquis, 2002, Fierens et al., 2015]. These circuits play a similar role as the grounded Bayesian and Markov network: they are used for efficient inference and learning.

The underlying learning algorithms and principles do not change. Both EM- and gradient-based algorithms have been used for learning [Sato, 1995, Gutmann et al., 2008, Fierens et al., 2015]. Some of these (such as PRISM and LeProbLog) have focused on learning from entailment, i.e., the examples are grounded facts that are assumed to have been randomly generated from the target program, others use the more traditional learning from interpretations setting.

For instance, the EM algorithm of PRISM [Kameya et al., 1999, Sato and Kameya, 2000, Kameya and Sato, 2000, Sato and Kameya, 2001] will, given an example, compute the set S of abductive hypotheses that entail the example from the possible proofs of the example. Such a hypothesis is a set of facts that is sufficient for the example to be proven. Therefore, each abductive hypothesis constitutes a completion of the example and can be weighted with the probability that it is true.

8.3.3 LEARNING WITH LATENT VARIABLES

In the extreme case, variables might not directly be observed but are rather inferred from other other variables. Such latent (hidden or unobserved) variables provide often a more sophisticated way to model relations.

Example 8.5 Consider the model of Fig. 3.1, but where the properties $i(S)$ and $d(C)$ are not observed, but are latent variables. Obviously, when these are unobserved variables, it may not be the intelligence of the students and the difficulty of the courses that are discovered by the learning; a learner will discover whichever categorizations best predict the observations. Inducing properties of individuals is equivalent to a (soft) clustering of the individuals. For example, if there are three values for $i(S)$, assigning a value to $i(S)$ for each student S is equivalent to assigning S to one of three clusters. The use of latent variables allows for much greater modeling than can be done with reference classes [Chiang and Poole, 2011].

Although the difficulty of the courses and the intelligence of the students are *a priori* independent, they become dependent once some grades are observed. For example, if students s

and c have taken a course in common (and the difficulty of the course is not observed), $i(s)$ and $i(c)$ are dependent, and if c and k have also taken another course in common, all three random variables are interdependent. It is this interdependence that makes inference difficult for binary observations [Jaeger and Van Den Broeck, 2012].

Some models can have many latent variables. For example, Koren et al. [2009] used hundreds of latent properties for predicting Netflix movie ratings. Since inference (for computing expected counts) is a key step in learning with latent variables, learning such models quickly becomes intractable.

8.4 STRUCTURE LEARNING OF PROBABILISTIC RELATIONAL MODELS

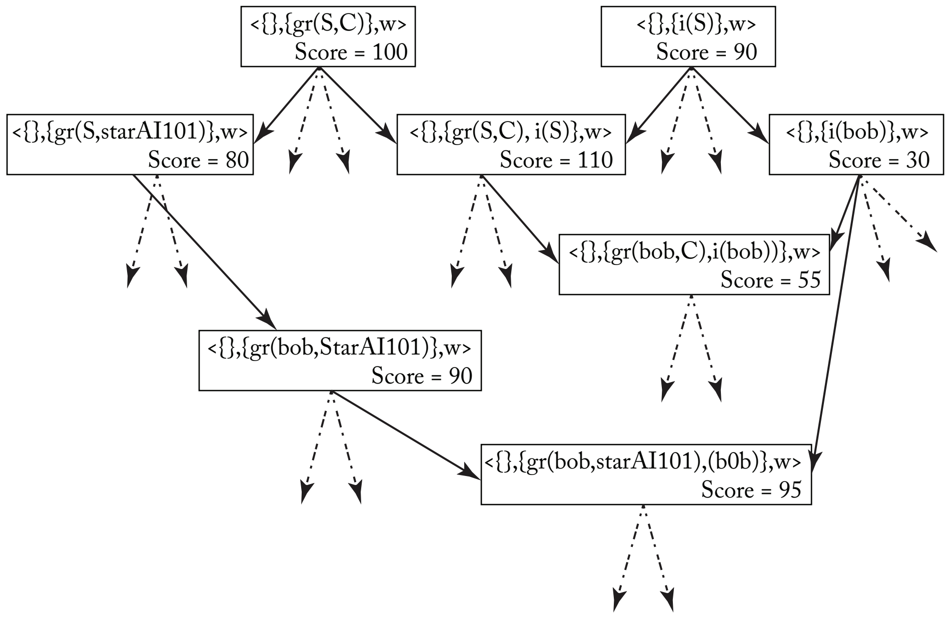
Learning the structure, which can be seen as determining which parameterized factors to include in a model, has also received considerable attention [Getoor et al., 2001a, Kok and Domingos, 2007, Mihalkova and Mooney, 2007, Domingos and Lowd, 2009]. The resulting learning approaches have been proven successful in many application domains as they can, e.g., employ autocorrelation [Jensen and Neville, 2002] but they may also discover misleading correlations [Jensen et al., 2003]. So far, there has been little theoretical analysis of statistical relational learning in general. Dhurandhar and Dobra [2012] extended, e.g., Hoeffding bounds to the relational classification setting, whereas Xiang and Neville [2011] outlines conditions under which learning from a single network will be asymptotically consistent and normal.

The task of **structure learning** aims at learning both the structure L and the parameters λ of the relational probabilistic model $H = (L, \lambda)$ from data. Often, further information is given as well. As in inductive logic programming, the additional knowledge can take various different forms, including a *language bias* that imposes restrictions on the syntax of L, and an *initial hypothesis* (L, λ) from which the learning process can start.

Learning the structure of probabilistic relational models requires a combination of techniques from inductive logic programming for structuring the search with evaluation scores that are based on graphical model learning techniques. Whereas ILP typically employs some $0 - 1$ score to evaluate hypotheses (e.g., the number of correctly covered examples), for learning probabilistic relational models one typically employs some probabilistic score such as (pseudo) likelihood (the likelihood of correctly covering the examples) with regularization to penalize complexity akin to structure learning in probabilistic graphical models. For statistical relational learning, there are essentially two issues:

1. defining what to optimize: use probabilistic scoring functions for assessing the quality of a probabilistic relational structure; and

2. designing and algorithm to do the optimization: use refinement operators to traverse the space of possible structures for the model.



<{},{gr(S,C)},w> Score = 100
<{},{i(S)},w> Score = 90
<{},{gr(S,starAI101)},w> Score = 80
<{},{gr(S,C), i(S)},w> Score = 110
<{},{i(bob)},w> Score = 30
<{},{gr(bob,C),i(bob))},w> Score = 55
<{},{gr(bob,StarAI101)},w> Score = 90
<{},{gr(bob,starAI101),(b0b)},w> Score = 95

Figure 8.2: Example of how relational structure learning proceeds as local search. Initially, the model has a parameterized factor $Gr(s, c)$ and then local refinements are performed to compute the score of each local refinement: parameterized random variables can be added to or deleted from the parameterized factor; logical variables can be replaces by constants; constants can be variablized; parameterized factor can be added to and deleted; and so on. Then the highest scoring one is chosen and the search continues after making that refinement.

8.4.1 A VANILLA STRUCTURE LEARNING APPROACH

With this in mind, the **vanilla structure learning** algorithm for probabilistic relational models might be sketched as a greedy hill-climbing search algorithm along the lines of structure learning algorithms for Bayesian networks sketched in Section 7.1.3. It is only that the refinements, see Fig. 8.2, are now relational that are more on the style used by the clausal discovery algorithm in Fig. 7.3.

Assuming at least one example, we take some initial model T_0, (e.g., *gr*, *i*, and *d* are independent), as starting point and compute the parameters maximizing some score such as the (pseudo) likelihood. Then, we use the refinement operators to compute neighbors of T_0. Just to recall, a **refinement operator** takes the current model, makes small, syntactic modifications to it, and returns a copy of the modified model. For Bayesian networks, typical refinements are adding, deleting, or flipping single edges [Heckerman, 1995]. For relational models, we instead add or

```
procedure $SRL(D:$ (set of) interpretations):
    $H$ := initial non-empty set of probabilistic clauses
    repeat until no improvement in score
        select probabilistic clause $c$ in $H$
        select refinement $d$ of $c$ such that score $H \setminus \{c\} \cup \{d\} >$ score $H$
                $H = H \setminus \{c\} \cup \{d\}$
    return $H$
```

Figure 8.3: A vanilla greedy SRL structure learning algorithm.

delete single literals to formulas, negate them, or instantiate or unify variables in them, cf. Fig. 8.2 and Section 7.2.2.

Example 8.6 For instance, we may hypothesize that the grade $gr(S, C)$ of student S in course C depends on the intelligence $i(S)$ of the student S. This can be done by adding $i(S)$ to the parameterized factor defining $gr(S, C)$.

This essentially corresponds to adding multiple edges in the underlying ground graphical model. Now, if the score for one of the neighbors, say H, is larger than the currently best model T_i, we take H as new current best model T_{i+1} and iterate. The process is continued until no further improvements in score are obtained, see, e.g., [De Raedt and Kersting, 2004, Kersting and De Raedt, 2008, De Raedt and Kersting, 2010]. This algorithm is presented as a pseduocode in Fig. 8.3.

As naively computing the score of each candidate T_i requires a parameter estimation step, which may be expensive when the data is only partially observable, one has devised variations of the above structure learning algorithm. One more efficient approach is **structural-EM** (**SEM**) [Friedman, 1998], which adapts the standard EM algorithm for structure learning. The key idea is that the expected counts are not computed anew for every structure that is proposed, but only after several iterations. This leads to improved efficiency.

By now, we are able to describe some of the key approaches to structure learning for relational probabilistic models.

8.4.2 PROBABILISTIC RELATIONAL MODELS

To arrive at a more detailed algorithm, the most important observation is that each of the relational graphical model formalisms employs a set of relational factors that act as a template for grounding out the probabilistic relational model. Each of these relational factors can be viewed as a clause on which the traditional refinement operators of inductive logic programming can be applied. Consider, for instance, the model in Fig. 3.1. It can be written as the rule

$$gr(S, C) \quad \leftarrow \quad i(S), d(C)$$

to which the standard ILP operators apply. Notice that $gr(S, C)$ would take the usual values (a, b, ...), and so the rule is a shorthand for a set of rules of the form $gr(S, C) = g \leftarrow i(S) = int, d(C) = diff$ where g, int and $diff$ are possible values for grade, intelligence and difficulty, cf. Kersting and De Raedt [2007]. Notice also that each application of a refinement operator on this rule acts a kind of macro at the level of the induced (ground) Bayesian network. Indeed, when a literal is added to the template, this corresponds to adding multiple edges to the induced Bayesian network on the data.

Now, depending on the formalism considered, different additional operations may be required or restrictions may need to be imposed.

Directed Probabilistic Logic Models

Relational upgrades of Bayesian networks such as PRMs, ICL, and BLPs have to ensure that upon grounding an acyclic Bayesian network is obtained. Therefore, as in Bayesian networks, special care must be taken to guarantee acyclicity, which usually requires extra tests on the candidate structures although some recent approaches try to avoid checking acyclicity, see, e.g., Schulte et al. [2012].

Furthermore, formalisms such as BLPs and PRMs also use aggregation or combining rules, which therefore must be taken into account by the refinement operators. For instance, PRMs can be represented by clauses such as $p(X) \; :- \; \tau_1(q(X, Y)), \tau_2(r(X, Z))$, where the τ_i denote different aggregate functions and where there is at most one clause defining a predicate (such as `p/1`). Structure learning with probabilistic relational models now occurs by refining this type of clauses [Friedman et al., 1999, Getoor et al., 2000, 2001b,a, Getoor, 2001]. Aggregated literals such as $\tau_1(q(X, Y))$ can be added to or deleted from clauses. With BLPs, one does not have to consider aggregators but the combining rules will be applied during parameter estimation which is typically done via EM algorithm as explained in the previous section.

Markov Logic Networks

For Markov Logic, as one works directly with full clausal logic, the standard inductive logic programming operators directly apply and there is no need to take into account acyclicity or combining rules. The initial approach to learning MLN structure simply operated in two steps—the first step was to use a ILP learner (the clausal discovery algorithm in Fig. 7.3) to learn a set of clauses and then fit the weights using any weight learning algorithm. This method ignores the interaction between the probabilistic and deterministic components of the model.

Consequently, there have been some advances to the vanilla learning approach for Markov Logic. The earliest work in this direction was due to Kok and Domingos [2005] who searched over the space of clauses and learned the weights of these clauses before scoring the candidate structure. They empirically demonstrated superior performance over simply learning the rules in one step and the weights in the next.

The same authors later developed a different approach [Kok and Domingos, 2009] to learning MLNs by viewing a given set of examples, i.e., a relational database, as a hypergraph. A hypergraph is a straightforward generalization of a graph, in which an edge can link any number of nodes, rather than just two. The constants appearing in an example are the nodes and ground atoms are the hyperedges. Each hyperedge is labeled with the predicate symbol of the corresponding ground atom. Nodes (constants) are linked by a hyperedge if and only if they appear as arguments in the hyperedge. Any path of hyperedges can be generalized into conjunctions of relational atoms by variablizing their arguments. They used relational path finding [Richards and Mooney, 1992] for learning MLNs by lifting these hypergraphs. Kok and Domingos [2010] extended this approach to start with a ground hypergraph and perform random walks to identify groups of nodes that can be clustered together. The clustering continues until no two clusters are similar in the hypergraph.

A similar bottom-up approach was earlier proposed by Mihalkova and Mooney [2007]. They identify ground atoms that share a constant and variablize the constants that are shared between ground clauses. The clauses are then constructed by combining the variablized nodes of the hypergraph (again found using relational path finding). Each path is turned into a set of conjunctions of atoms for which we estimate the weights. Van Haaren et al. [2015] showed how to exploit exact lifted inference for learning the structure of MLNs. Recently, a boosted approach to learning MLNs has been developed [Khot et al., 2011] and we cover this later in the chapter.

Probabilistic Programs

The problem of learning the structure of probabilistic logic programs differs from the frameworks previously discussed in this section in that the learning problem need not be related to that of learning graphical models, Section 7.2.5. However, when considering the learning from interpretations setting, the same principles appply. For instance, Bellodi and Riguzzi [2015] proposed recently to perform a beam search in the space of probabilistic clauses and a greedy search in the space of theories using the log likelihood of the data as the guiding heuristics. Di Mauro et al. [2015] considered learning the structure of a probabilistic logic program as a multi-armed bandit problem. The resulting LEMUR approach relies on the Monte-Carlo tree search UCT algorithm that combines the precision of tree search with the generality of random sampling.

On the other hand, learning from entailment imposes an additional requirement on the induced programs. Basically, the examples then become ground that must be (probabilistically) entailed by the target (probabilistic) logic program. More formally, when ommitting the parameters (and the probabilities) from the logic program, the resulting program H must entail each of the examples e. Solving the general structure learning problem for probabilistic programs thus involves applying a refinement operator at the *theory* level (i.e., considering multiple predicates) under entailment. This problem has been studied in inductive logic programming under the term theory revision [Shapiro, 1983, De Raedt, 1992, Wrobel, 1996] and is known as a hard problem. This may explain why this most general form of learning has not yet received a lot of attention and why—to the best of our knowledge—there exist no structure learning algorithms for Sato's PRISM yet. So far, there have been approaches to learning a single predication only, both for what concerns stochastic logic programs [Muggleton, 2002] (a formalism based on the probability of proofs inspired by probabilistic grammars) and ProbLog [De Raedt et al., 2015]. The later approach essentially upgrades the inductive logic programming problem to probabilistic programming in that each example is given a target value and an adaptation of the standard rule learning algorithm finds a set of ProbLog rules that tries to best approximate these target values.

8.4.3 BOOSTING

Another recent advancement is triggered by the insight that finding many rough rules of thumb of how to change our probabilistic relational models locally can be a lot easier than finding a single, highly accurate local model.

Consider for example relational dependency networks [Neville and Jensen, 2007]. Instead of learning the conditional probability distribution associated with each predicate using relational tree learning in single shot manner, one can represent it as a weighted sum of regression models grown in a stage-wise optimization using gradient **boosting** [Natarajan et al., 2012b]. This corresponds to a functional gradient approach and was originally proposed for training conditional random fields for labeling relational sequences [Gutmann and Kersting, 2006] and for learning relational policies [Kersting and Driessens, 2008].

The key idea in this line of work is to represent each conditional distribution as a set of relational regression trees (RRT) [Blockeel and De Raedt, 1998], see Fig. 8.4 for an example of RRT. The goal is to predict if person A is *advised by* person B given the properties and relations of people at a university. The second branch from the left states that if B is a professor, A is not a professor, A has more than 1 publication, and has more than 1 common publication with B, then the regression value is 0.05. These regression values are then exponentiated and normalized.

To learn this set of RRTs for each conditional distribution, the learning approach was adapted based on the gradient boosting technique and was called relational functional-gradient boosting (RFGB) [Natarajan et al., 2012b, 2015]. Assume that the training examples are of the form $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, N$ and $y_i \in \{1, \ldots, K\}$. Let us use $\mathbf{x}$ to denote the set of non-target predicates (features) and y_i to denote the current target predicate. Then the goal is to fit a model $P(y|\mathbf{x}) \propto e^{\psi(y,\mathbf{x})}$. The key idea is to compute the gradient (weight) for each example separately and fit a regression tree over all the weighted examples. This set of local gradients will approximate the global gradient. The functional gradient of each example $\langle \mathbf{x}_i, y_i \rangle$ w.r.t likelihood $(\psi(y_i = 1; \mathbf{x_i}))$ is

$$\frac{\partial \log P(y_i; \mathbf{x_i})}{\partial \psi(y_i = 1; \mathbf{x_i})} = I(y_i = 1; \mathbf{x_i}) - P(y_i = 1; \mathbf{x_i}), \tag{8.1}$$

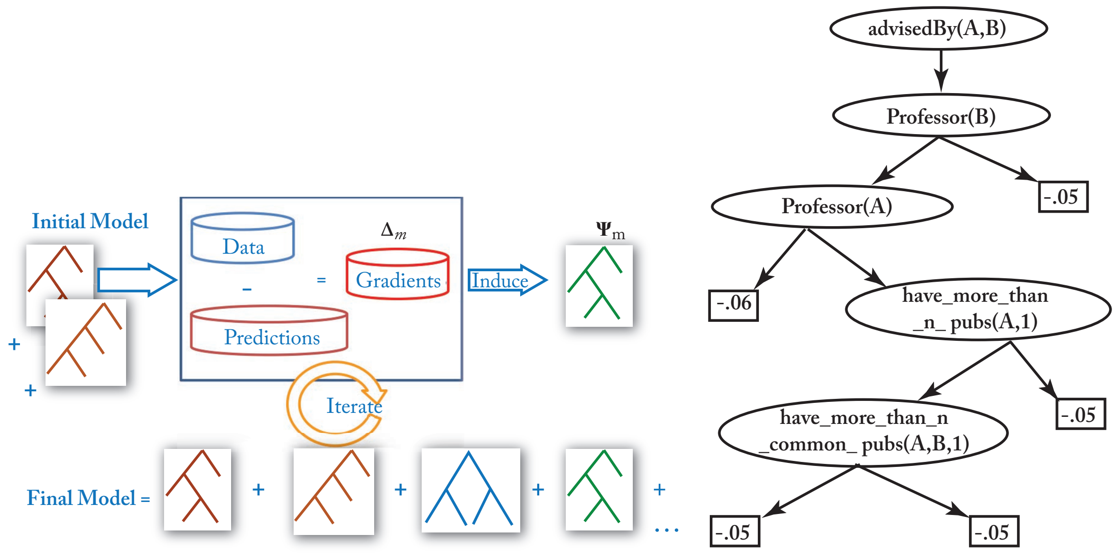


Figure 8.4: (Left) Relational functional gradient boosting. This is similar to the standard gradient-boosting where trees are induced in stage-wise manner. At every iteration, the gradients are computed as the difference between observed and predicted probabilities of each example and a new regression tree is fitted to these examples. (Right) Example of a relational regression tree. The goal is to predict if a person$A$ is advisedBy B (where A and B are logical variables). At each node a (set of) predicate(s) is evaluated. The leaves denote regression values that are exponentiated and normalized to obtain the probabilities.

where I is the indicator function that is 1 if $y_i = 1$ and 0 otherwise. The expression is simply the adjustment required to match the predicted probability with the true label of the example. If the example is positive and the predicted probability is less than 1, this gradient is positive indicating that the predicted probability should move toward 1. Conversely, if the example is negative and the predicted probability is greater than 0, the gradient is negative, driving the value the other way. They use RRTs to fit the gradient function for every training example.

Each RRT can be viewed as defining several new feature combinations, each corresponding to one of the paths from the root to a leaf. The resulting potential functions from all these different RRTs still have the form of a linear combination of features but the features can be quite complex. This idea is illustrated in Fig. 8.4. The benefits of a boosted learning approach are manifold. First, being a nonparametric approach the number of parameters grows with the number of training episodes. In turn, interactions among random variables are introduced only as needed, so that the potentially large search space is not explicitly considered. Second, such an algorithm is fast and straightforward to implement. Existing off-the-shelf regression learners can be used to deal with propositional, continuous, and relational domains in a unified way. Third, it learns the structure and parameters simultaneously, which is an attractive feature as learning probabilistic relational models is computationally quite expensive. For an implementation as well ass further material on how to use it, we refer to `http://pages.cs.wisc.edu/~tushar/rdnboost/index.html`.

While originally developed for relational dependency networks [Natarajan et al., 2012b], this work was later extended to learning MLNs by simply optimizing the pseudo-likelihood [Khot et al., 2011] and to learning both RDNs and MLNs in the presence of hidden data [Khot et al., 2015a]. Based on Ravkic et al. [2015], who recently showed how to learn hybrid relational dependency networks, this may also be extended to hybrid domains. Using the connection between RDNs and MLNs, one could also learn MLNs by first learning a (parametrized) Bayesian network (BN) and then transform the Bayesian network to an MLN [Khosravi et al., 2012, Schulte and Khosravi, 2012, Schulte et al., 2014].

8.5 BAYESIAN LEARNING

Whereas we can find the most likely model given the data [Sato and Kameya, 1997, Getoor et al., 2001a] using the structure learning approach sketched above, we may also take a Bayesian perspective and average over all models. From a Bayesian point of view, learning is just a case of inference: we condition on all of the observations (all of the data), and determine the posterior distribution over some hypotheses or any query of interest. Starting from the work of Buntine [1994], there has been considerable work in using relational models for **Bayesian learning** [Jordan, 2010]. This work uses parameterized random variables (or the equivalent plates) and the probabilistic parameters are real-valued random variables (perhaps parameterized). Dealing with real-valued variables requires sophisticated reasoning techniques, often in terms of MCMC and stochastic processes. Although these methods use relational probabilistic models for learning, the representations learned are typically not relational probabilistic models although the approach is also popular for learning functional probabilistic programs [Pfeffer, 2007, Goodman et al., 2008]. It is still an open challenge to bring these two threads together, mainly because of the difficulty of inference in these complex models.

One challenge of learning is that we may want an agent to learn general hypotheses before it knows the individuals it will encounter, and so before the agent knows the random variables. When the agent encounters and observes the new individuals, it can learn about these individuals (as in Section 8.1).

PART IV

Beyond Probabilities

CHAPTER 9
Beyond Basic Probabilistic Inference and Learning

So far, we have shown how to combine logic and probabilities for standard inference tasks such as computing marginals, MAP, and learning the structure of relational probabilistic models. In many real-world applications, however, the problem formulation does not fall neatly into the pure probabilistic inference case. The problem may for example have a combinatorial component, hence, taking it outside the scope of standard inference tasks.

As we will illustrate now, many—if not all—ideas and concepts underlying StarAI go beyond standard inference tasks and actually cover major parts of the AI spectrum.

9.1 LIFTED SATISFIABILITY

Our first illustration is the classical **Boolean satisfiability** (**SAT**) problem itself already considered in Chapter 5. To recap, SAT consists of a formula F representing a set of constraints over n Boolean variables, which must be set so as to satisfy all constraints. It is one of the most—if not the most—studied problem in computer science and AI, and other NP-complete problems can be reduced to it and, hence, "compilation to SAT" is a powerful paradigm for solving computer science and AI problems and has applications in several important areas such as automated deduction, cryptanalysis, machine learning, modeling biological networks, hardware and software verification, and planning, among others.

In turns out that message-passing algorithm for solving SAT auch as namely **warning propagation** (WP) and **survey propagation** (SP) [Mézard et al., 2002], can be viewed as a form of belief propagation [Braunstein and Zecchina, 2004] and, hence, are amenable to lifting, too. The basic step is to view a Boolean formula given in **Conjunctive Normal Form** (**CNF**)—a conjunction of Boolean literals—as factor graph.

Example 9.1 Consider the following CNF, consisting of three variables, three clauses, and seven literals: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3)$. Every CNF can be represented as a factor graph [Kschischang et al., 2001]. This is illustrated in Fig. 9.1 (left) where we use a dashed line between a variable and clause whenever the variable appears negated in a clause, i.e., $s_{i,j} = -1$, otherwise a full line.

Now, we run **color-passing**—as introduced in Chapter 6 for lifting BP—to compute a lifted CNF factor graph and then run a modified warning propagation resp. survey propagation

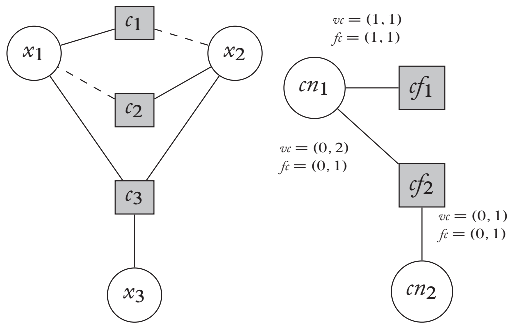


Figure 9.1: (Left) The CNF $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee x_3)$ represented as factor graph. Circles denote variables nodes and squares factors. A dashed line indicates that the variable appears negated in a clause; otherwise we use a full line. (Right) The lifted CNF factor graph resulting from runnning color-refinement.

on the lifted graph. The lifted messages for WP and SP are a little bit involved and to not add to the understanding of lifted inference. Consequently, we refer the reader to Hadiji et al. [2010]. The experimental results in Hadiji et al. [2010] show that lifted warning propagation and survey propagation can sent significantly less many messages as their propositional counterparts.

9.2 ACTING IN NOISY RELATIONAL WORLDS

Many real-world problems can be cast as sequential decision making under uncertainty. Consider a simple example in a logistics domain where an agent delivers boxes. The agent can take three types of actions: to load a box on a truck, to unload a box from a truck, and to drive a truck to a city. However, the effects of actions may not be perfectly predictable. For example, its gripper may be slippery so load actions may not succeed, or its navigation module may not be reliable and it may end up in a wrong location. This uncertainty compounds the already complex problem of **acting optimally**, i.e., of planning a course of action to achieve some goals or to maximize rewards.

Markov decision processes (MDPs) [Bellman, 1957] model an agent interacting with the world and have become the standard model for sequential decision making under uncertainty. The agent can fully observe the state of the world and takes actions so as to change the state. In doing that, the agent tries to optimize a measure of the long term reward it can obtain by taking actions. The classical representation and algorithms for MDPs [Puterman, 1994], however, require the enumeration of the state space. Assume, e.g., that there are four trucks, three boxes, and the goal is to have a box in Paris, but it does not matter which box is in Paris. We need to encode every possible instantiation of the relations in the domain, e.g., "Box1 in Paris," "Box 2 in Paris," "Box 1 on Truck 1," "Box 2 on Truck 1," and so on, to form the state-space, and the action space expands in the same way. The goal becomes a disjunction over different instances stating "Box 1 in Paris

$\vee$ Box 2 in Paris $\vee$ Box 3 in Paris $\vee$ Box 4 in Paris." Thus, although arguably a simple instance, we get already a rather large representation. More importantly, we lose the structure implicit in the relations. Each instance is composed of these same basic building blocks, but they differ in terms of where the trucks are, whether a box is on a truck or not, etc. We would like the agent to make use of this structure in terms of computing plans that apply across multiple scenarios. This is the main motivation behind relational or first-order logical MDPs. They describe domain objects and relations among them, and can use quantification in specifying objectives.

There is actually a large body of work on relational representations of stochastic actions and noisy sensing. The initial work in this area was on representations, in terms of the event calculus [Poole, 1997] or the situation calculus [Poole, 1998, Bacchus et al., 1999]. Later work has concentrated on how to do planning with such representations either for the fully observable case [Boutilier et al., 2001, Kersting et al., 2004, Sanner and Boutilier, 2009, Wang et al., 2008, Van den Broeck et al., 2010] or the partially observable case [Wang and Khardon, 2010, Sanner and Kersting, 2010b] at a lifted level. In essence, they run **symbolic dynamic programming** (SDP)—a generalization of the dynamic programming technique for solving propositional Markov decision processes, see, e.g., Bellman [1957]—exploiting the symbolic structure in the solution of relational and **first-order logical Markov decision processes** through a lifted version of dynamic programming [Sanner and Kersting, 2010a].

SDP constructs a minimal logical partition of the state space required to make the distinctions necessary to give current and future rewards for various lookaheads. To give a flavor, reconsider the logistics domain.

Example 9.2 We receive a reward of 10 if a box B is in Paris and otherwise 0:

$$10.0 : \quad \exists B\ boxIn(B, paris),$$

where B is a logical variable. This says that we receive the rewards whenever there is at least one box in Paris. If there are more than one box in Paris, we still get only a reward of 10.

The actions the agent can perform include load, unload, and drive. To encode them, one can use probabilistic STRIPS-like planning operators [Fikes and Nilsson, 1971, Pasula et al., 2007]

Example 9.3 For instance, the action to unload a box B from a truck T in city C can be defined as:

$$\begin{aligned} \forall B, T, C \quad unload(B, T, C) :: boxOn(B, T) \wedge truckIn(T, C) \rightarrow & \\ 0.9 : boxIn(B, C) \wedge \neg boxOn(B, T) & \\ 0.1 : boxOn(B, T) \wedge truckIn(T, C) & \end{aligned}$$

where the signs of the literals denote the add and delete effects of the actions. The add effects describe what is added to the world as an effect of the action, delete effects what is deleted from the world. So with the unload action above, when there is a truck T in a city B and there is a box B on

T, then unloading B succeeds with probability 0.9 and we add $boxIn(B, C)$ and delete $boxOn(B, T)$ to the world. With probability 0.1 the action fails and we change nothing. Or in a more concrete situation, if there is a box box_1 on a truck $truck_1$ in a city $paris$, then with probability 0.9 we remove $boxOn(box_1, truck_1)$ from the current state and add $boxIn(box_1, paris)$ when performing action $unload(box_1, truck_1, paris)$.

Now the domain dynamics and rewards are described compactly and abstractly using the relational notation. In turn, the agent can compute a logically parametrized plan that can be applied to any situation. For instance, it is possible to infer that in order to get box B to be in city C, the agent drives a truck T to B, loads B onto T, drives T to c, and finally unloads B.

This compact and abstract policy is computed by regression. In our running example, let $Value(t)$ be the t stages-to-go value function (i.e., is the discounted reward obtained in the next t time steps). Intuitively, the **value function** of a policy encodes the expected sum of discounted rewards accumulated while executing that policy when starting from state s [Bellman, 1957]. Of course, $Value(0)$ is the immediate reward as defined above. Now, we regress all preceding states from all states in $Value(0)$ for all actions. For example, the outcome of a succeeding unload action can lead to the goal state from any state where

$$\exists B, T\; truckIn(T, paris) \wedge boxOn(B, T) \wedge \neg boxIn(B, paris) \ .$$

This is computed by matching the outcome in all possible ways with a state in the current value function. In the failing case of the unload action, we are not reaching the goal state, that is, we have to treat

$$\exists\, B, T \neg\Big(truckIn(T, paris) \wedge boxOn(B, T)\Big) \wedge \neg boxIn(B, paris)$$

separately. Now, we compute the expected rewards of the outcomes and combine them into a value function.

Example 9.4 The one stage-to-go value function for the goal $\exists B\; boxIn(B, paris)$ is:

$$\begin{aligned} Value(1) \equiv\;\; & 19 \;:\; \exists\, B\; boxIn(B, paris) \\ & \;\;9 \;:\; \exists\, B, T\; truckIn(T, paris) \wedge boxOn(B, T) \wedge \neg boxIn(B, paris) \\ & \;\;0 \;:\; \forall\, B, T \neg\Big(truckIn(T, paris) \wedge boxOn(B, T)\Big) \wedge \neg boxIn(B, paris) \ . \end{aligned}$$

As one can see, the first logical partition is just the reward partition. The second checks whether there is a loaded truck T in paris. If so, we can unload the box B and fall back into the first partition. Otherwise, we do not receive any reward as we cannot reach the goal in one step.

While the technical details of this are beyond the scope of the book, one should note that the operations are exactly the lifted versions of the traditional dynamic programming solution to Markov decision processes and are called **first-order decision-theoretic regression** and symbolic maximization [Sanner and Boutilier, 2007, 2009]. After sufficient iterations, the t-stages-to-go value function converges.

The two stages-to-go value function for the goal $\exists B\, boxIn(B, paris)$ can be computed to be:

$$
\begin{aligned}
Value(2) \equiv\ & 27.1 &&: \exists\, B\ boxIn(B, paris)\\
& 17.1 &&: \exists\, B, T\ truckIn(T, paris) \wedge boxOn(B, T) \wedge \neg boxIn(B, paris)\\
& 8.1 &&: \exists\, B, C, T\ boxOn(B, T) \wedge truckIn(T, C) \wedge\\
& && \quad \neg\Big(truckIn(T, paris) \wedge boxOn(B, T)\Big) \wedge \neg boxIn(B, paris)\\
& 0 &&: \forall\, B, C, T\ \neg\Big(boxOn(B, T) \wedge truckIn(T, C)\Big) \wedge\\
& && \quad \neg\Big(truckIn(T, paris) \wedge boxOn(B, T)\Big) \wedge \neg boxIn(B, paris)\ .
\end{aligned}
$$

While the exact representation and SDP solution differ among the variant formalisms, they all share the same basic logical representation of rewards, probabilities, and values: the state and action abstraction in the value and policy representation are afforded by the first-order specification and solution of the problem. That is, this solution does not refer to any specific set of domain individuals such as $truck1, truck2, box1, \ldots$, but rather it provides a lifted solution using placeholders such as B and T where possible. While classical dynamic programming techniques, which enumerate all states and actions, could never solve these problems for large domain instantiations, the lifted solution is quite simple due to the power of logic, i.e., state and action abstraction.

Since the basic symbolic dynamic programming approach, a variety of exact algorithms have been introduced to solve MDPs with relational (RMDP) and first-order (FOMDP) structure. Examples are *first-order value iteration* (FOVIA) [Hölldobler and Skvortsova, 2004, Karabaev and Skvortsova, 2005] and the *relational Bellman algorithm (ReBel)* [Kersting et al., 2004]. They are value iteration algorithms for solving RMDPs. In addition, *first-order decision diagrams (FODDs)* have been introduced to compactly represent case statements and to permit efficient application of symbolic dynamic programming operations to solve RMDPs via value iteration and policy iteration [Wang et al., 2008]. All of these algorithms have some form of guarantee on convergence to the (ϵ-)optimal value function or policy. Furthermore, a class of linear-value approximation algorithms have been introduced to approximate the value function as a linear combination of weighted basis functions. *First-order approximate linear programming (FOALP)* [Sanner and Boutilier, 2009] directly approximates the FOMDP value function using a linear program. Other heuristic solutions for instance induce rule-based policies from sampled experience in small-domain instantiations of RMDPs and generalize these policies to larger domains [Fern et al., 2003]. In a similar vein, Gretton and Thiebaux [2004] used the action regression operator in the situation calculus to provide the first-order hypothesis space for an inductive policy learning algorithm. Recently, Lang and Toussaint [2009] and Joshi et al. [2010] showed that successful planning typically involves only a small subset of relevant individuals (or states) and how to make use of this fact to speed up symbolic dynamic programming significantly. Powell [2010] described a framework for reasoning in such domains by counting over the number of individuals. Van den Broeck et al. [2010] showed how to use probabilistic programming for making decisions and Nitti et al. [2015] extended this to planning in hybrid MDPs. Recently, Hescott and Khardon [2014] started to investigate the complexity of manipulating some of the data structures used for symbolic dynamic programming.

In general, solvers for **relational MDPs** have been successfully applied in decision-theoretic planning domains such as BlocksWorld, BoxWorld, ZenoWorld, Elevators, Drive, PitchCatch, and Schedule as typically used within the International Planning Competition, see, e.g., Fern et al. [2006], Sanner and Boutilier [2009]. The FOALP system [Sanner and Boutilier, 2009] was runner-up at the probabilistic track of the 5th International Planning Competition (IPC-6). Related techniques have been used to solve path planning problems within robotics and instances of real-time strategy games, Tetris, and Digger [Driessens and Dzeroski, 2002, Lang and Toussaint, 2010, Sarjant et al., 2011, Lang et al., 2012].

In many situations, however, the agent does not have knowledge about the underlying (relational) (PO)MDP given. In turn, the agent has to learn what to do through trial-and-error interactions with an uncertain, relational environment. This is called **relational reinforcement learning** [Dzeroski et al., 2001, Tadepalli et al., 2004, van Otterlo, 2009]. To deal with the obvious "curse of dimensionality" encountered—there are simply too many potential states—a number of model-free [Dzeroski et al., 2001, Driessens and Ramon, 2003, Driessens et al., 2006, Driessens and Dzeroski, 2005, Ramon et al., 2007, Kersting and Driessens, 2008] as well as model-based [Pasula et al., 2007, Lang and Toussaint, 2010, Lang et al., 2012] relational reinforcement learning approaches have been developed. The core idea is to lift classical reinforcement learning approaches by using relational machine learning techniques that can abstract from individuals using logical queries. Relational learners such as relational regression trees, graph kernels, relational gradient boosting, and inductive logic programming methods are used either to learn relational value functions, relational policies, or even R(PO)MDPs.

For instance, we may follow the **boosting** idea already discussed for learning relational probabilistic models. Triggered by the idea that finding many rough rules of thumb of how to change the way to act can be a lot easier than finding a single, highly accurate policy, one can represent a policy—mapping relational states to actions—as a weighted sum of relational regression trees grown in an stage-wise optimization in order to maximize the expected return [Kersting and Driessens, 2008]. Each regression tree can be viewed as defining several new feature combinations, one corresponding to each path in the tree from the root to a leaf. However, rather than using attribute-value or threshold tests in an inner node of the tree, a relational regression tree employs logical queries such as $boxIn(B, C)$. This way, interactions among (sets of) states and actions are introduced only as needed, so that the potentially infinite search space is not explicitly considered.

A key insight for relational reinforcement learning is that the inherent generalization of learnt knowledge in the relational representation has profound implications on the **exploration** strategy: what in a propositional setting would be considered a novel situation and worth exploration may in the relational setting be an instance of a well-known context in which exploitation is promising [Lang et al., 2010].

Example 9.5 Consider household robots, which just need to be taken out of their shipping boxes, turned on, and then do some cleaning work. This "robot-out-of-the-box" has inspired research in robotics as well as in machine learning and artificial intelligence. Without a compact knowledge representation that supports abstraction by logical placeholders, and generalization of previous experiences to the current state and potential future states, however, it seems to be difficult—if not hopeless—to explore a new home in reasonable time. There are simply too many individuals (items/objects) a household robot may deal with such as doors, plates, boxes and water-taps. With a relational model at hand, however, the robot can learn about how doors work, how taps work, and how to wash dishes, even before it is delivered, and can then observe the particular individuals (items/objects) in the house, and continue learning. After having opened one or two water taps in the kitchen of the new owners, e.g., to fill the sink with water, the household robot can expect other water-taps to behave similarly. Thus, the priority for exploring water-taps in kitchens in general should be reduced and not just the one for those ones that were tested. Moreover, our information gathered about these water-taps should potentially transfer to water-taps in laundries, since we have also learned something about water tapes in general. Without extensive feature engineering this would be difficult—if not impossible—in a propositional setting. We would simply encounter a new and therefore unexplored situation.

9.3 RELATIONAL OPTIMIZATION

The use of weighted model counting for probabilistic inference, lifted SAT as well as the linear programming approach for acting optimally in relational worlds by Sanner and Boutilier [2009] suggest that instead of looking at AI through the glasses of probabilities over possible worlds, we may also approach it using optimization. That is, we have a preference relation, i.e., some objective function over possible worlds, and we want a best possible world according to the preference. Consider for example a typical data analyst solving a machine learning problem for a given dataset. She selects a model for the underlying phenomenon to be learned (choosing a learning bias), formats the raw data according to the chosen model, tunes the model parameters by minimizing some objective function induced by the data and the model assumptions, and may iterate the last step as part of model selection and validation.

This is an instance of the declarative "Model + Solver" paradigm that was and is prevalent in AI [Geffner, 2014], natural language processing [Rush and Collins, 2012], machine learning [Sra et al., 2011], data mining [Guns et al., 2011], and some of the inference techniques we outlined earlier.

> Instead of outlining how a solution should be computed, we specify what the problem is in terms of some high-level modeling language and solve it using general solvers.

In particular, NLP applications witnesses a growing need for relational mathematical modeling languages [Riedel and Clarke, 2006, Yih and Roth, 2007, Clarke and Lapata, 2008, Martins et al., 2009, Riedel et al., 2012, Cheng and Roth, 2013, Kordjamshidi et al., 2015]

As an didactic example for illustration only, consider the **maximum flow** problem.

Example 9.6 The maximum flow problem involves finding a feasible flow through a single-source, single-sink flow network that is maximal [Ahuja et al., 1993]. That is, given a finite directed graph $G(V, E)$ in which every edge $(u, v) \in E$ has a non-negative capacity $c(u, v)$, and two vertices s and t called source and target, our goal is to maximize the flow $\sum_{v \in V/\{s,v\}} f(s, v)$. In order to be a valid flow, f must be subjected to the following constraints:

$$f(u, v) \leq c(u, v),\ f(u, v) \geq 0,\ \text{and}\ \sum_{w \in V/\{s,t\}} f(u, w) = \sum_{w \in V/\{s,t\}} f(w, u)\ .$$

The latter constraint asserts that the incoming flow equals the outgoing flow for internal vertices, that is, no "goods" are created or destroyed in nodes that are not the source or the sink. Since the objective and the constraints are all linear, this is a **linear program** (LP) [Bertsimas and Tsitsiklis, 1997], and to build it for a given network, we first introduce variables to represent flow over each edge of the network. Then, we formulate the capacity constraints and conservation constraints depending on the network structure. Finally, the objective function of the LP is the value of the total flow in the network.

As one can see, the general definition of the flow LP can be separated from the specification of the network at hand and, in turn, be applied to different networks. Very much like for relational probabilistic models, it is therefore sensible to encode LPs relationally [Kersting et al., 2015]. For an implementation as well as further material on how to use the resulting **relational linear programs**, we refer to `http://www-ai.cs.uni-dortmund.de/weblab/static/RLP/html/index.html`.

A **relational linear program** for the flow problem is shown in Fig. 9.2. It directly codes the flow constraints, concisely captures the essence of flow problems, and illustrates nicely that linear programming can be considered to be highly relational in nature. To do so, we intuitively follow the idea of par-factors already used to specify relational probabilistic models. Specifically, **LP predicates** such as `flow/2` and **LP atoms** define logically parametrized LP variables and parameters here the flows between nodes. The range of LP predicates are the real numbers; for instance the specific flow between node `f` and `t` could take the value 3.7, i.e., `flow(f, t) = 3.7`. Additionally, they can be subject to optimization:

```
1 # declarations of LP predicates
2 var flow/2, outflow/1, inflow/1;
```

denotes values to be determined by the LP solver. To encode the objective (line 4) and the constraints (lines 5–17) of the flow problem, we use logically parameterized algebraic expressions, or **par-expressions** in short. Instead of logical operators we use arithmetic aggregators (indexed over logical queries) and arithmetic operators. In our flow running example, the objective reads

```
# declarations of LP predicates
var flow/2, outflow/1, inflow/1;
#objective
maximize: sum {X in source(X)} {outflow(X)};
#auxiliar variables capturing the outflow of nodes
subject to forall{X in vertex(X)}:
   {outflow(X) = sum {Y in edge(X,Y)} {flow(X,Y)} };
#auxiliar variables capturing the inflow of nodes
subject to forall{Y in vertex(Y)}:
   {inflow(Y) = sum {X in edge(X,Y)} {flow(X,Y)} };
#conservation of flow
subject to forall{X in vertex(X), not source(X), not sink(X)} :
   {outflow(X) - inflow(X) = 0};
#capacity bound
subject to forall {X,Y in edge(X,Y)} : {cap(X,Y) - flow(X,Y) >= 0};
#no negative flows
subject to forall {X,Y in edge(X,Y)} : {flow(X,Y) >= 0};
```

Figure 9.2: Relational linear program encoding the maximal flow problem. For details we refer to Kersting et al. [2015].

```
#objective
maximize: sum {X in source(X)} outflow(X);
```

This says that we want to maximize the outflows for all source nodes. The constraints for the auxiliary LP predicate `outflow/2` is

```
#auxiliar variables capturing the outflow of nodes
subject to forall{X in vertex(X)}:
   {outflow(X) = sum {Y in edge(X,Y)} {flow(X,Y)} };
```

Since the logical variable `Y` is bound by the summation for `outflow/1` and the logical variable `X` by the all-quantification, this says that there will be one equality expression as constraint per node summing over all flows of the outgoing edges. Likewise, we can define the auxiliary LP predicate `inflow/2` summing over all ingoing edges:

```
#auxiliar variables capturing the inflow of nodes
subject to forall{Y in vertex(Y)}:
   {inflow(Y) = sum {X in edge(X,Y)} {flow(X,Y)} };
```

The remaining constraints are defined in a similar fashion:

```
#conservation of flow
subject to forall{X in vertex(X), not source(X), not sink(X)} :
   {outflow(X) - inflow(X) = 0};
#capacity bound
subject to forall {X,Y in edge(X,Y)} : {cap(X,Y) - flow(X,Y) >= 0};
#no negative flows
subject to forall {X,Y in edge(X,Y)} : {flow(X,Y) >= 0};
```

Indeed, several predicates such as `vertex/1` and `source/1` are not defined in the relational flow LP. In such cases, we assume the predicate to be defined in a logical knowledge base:

```
cap(s,a) = 4.  cap(s,b) = 2.  cap(a,c) = 3.  cap(b,c) = 2.
cap(b,d) = 3.  cap(c,b) = 1.  cap(b,t) = 2.  cap(d,t) = 4.
```

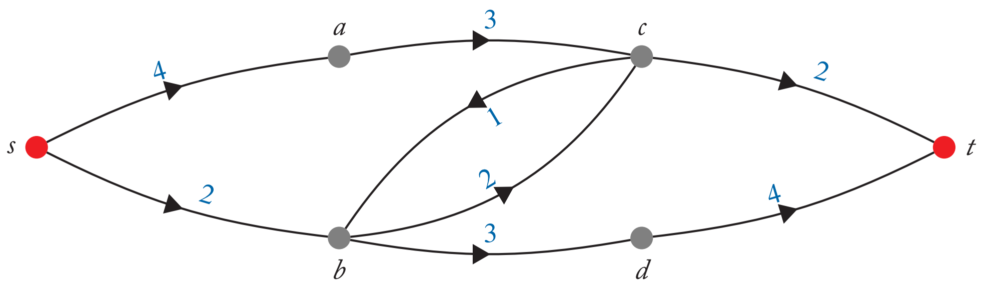


Figure 9.3: A graph for a particular instance of the flow problem. The node *s* denotes the source, and *t* the target. The numbers associated with the edges are the flow capacities.

```
edge(X,Y) :- cap(X,Y).

vertex(X) :- edge(X,_).
vertex(X) :- edge(_,X).

source(s).
target(t).
```

where `cap(s, a) = 4` (defining the maximum amount of flow that can pass through the edge) is short-hand notation for `cap(s,a,4)` and `cap(X, Y)` for `cap(X,Y,_)`, where we use an anonymized variable'_'.

Together with the logical knowledge base, the relational LP can be shown to induce an LP as long as the logical querie produces finite answer sets using the standard **grounding** techniques. This LP in turn can be solved using any existing LP solver. In our flow example, the resulting LP looks as follows (for the sake of readability, only some of the groundings are shown):

```
maximize: flow(s,a) + flow(s,b);
subject to outflow(a) = flow(a,c);
subject to outflow(b) = flow(b,c)+flow(b,d);
...
subject to inflow(a) = flow(s,a);
subject to inflow(b) = flow(s,b)+ flow(c,b);
...
subject to outflow(a) - inflow(a) = 0;
...
subject to 4 - flow(s, a) >= 0;
subject to 2 - flow(s, c) >= 0;
...
subject to flow(s, a) >= 0;
...
```

Thus, it is sufficient to change the LogKB—e.g., we may encode a different network with different capacity values—to induce different flow LPs; we do not have to touch the "LP logic" anymore. In turn, relational mathematical programming—actually nothing prevents us from specifying quadratic or other mathematical programs in the same fashion—allows a more intuitive

$$\begin{aligned} \underset{[x,y,z]^T \in \mathbb{R}^3}{\text{minimize}} \quad & 0x + 0y + 1z \\ \text{subject to} \quad & \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \leq \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} \end{aligned}$$

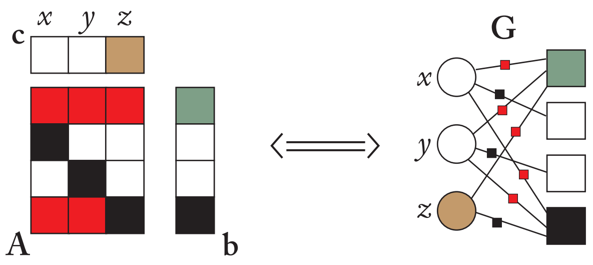


Figure 9.4: Construction of the coefficient graph G_L of the linear program L in the top panel. On the left-hand side, the coloring of the LP is shown. This turns into the colored coefficient graph shown on the right-hand side.

representation of optimization problems over relational domains where we have to reason about a varying number of objects and relations among them, without enumerating them.

Moreover, as shown in Kersting et al. [2015], linear programs can also be lifted, i.e., compressed using **color refinement**, as already discussed in Chapter 6. We will not go into the mathematical details of **lifted linear programming** but just present the the graphical representation G_L of the linear program $L = (A, b, c)$, called the **coefficient graph** of L, on which one runs color refinement.

To construct the coefficient graph G_L, we add a vertex to G_L for every of the m constraints and n variables in the linear program L. Then, we connect a constraint vertex i and variable vertex j *if and only if* $\mathbf{A}_{ij} \neq 0$. Furthermore, we assign colors to the edges $\{i, j\}$ in such a way that $color(\{i, j\}) = color(\{u, v\}) \iff \mathbf{A}_{ij} = \mathbf{A}_{uv}$. Finally, to ensure that **c** and **b** are preserved by any automorphism we find, we color the vertices in a similar manner, i.e., for row vertices i, j $color(i) = color(j) \iff b_i = b_j$ and $color(u) = color(v) \iff \mathbf{c}_u = \mathbf{c}_v$ for column vertices. We must also choose the colors in a way that no pair of row and column vertices share the same color; this is always possible. One can verify that an coloring of G_L yields coloring of L. From the complexity of color refinement and the fact that our graphical construction grows only linearly with the size of the LP, it follows that LPs can be lifted, i.e., compressed in quasilinear time in terms of the number of variables and constraints.

Several experimental results have demonstrated empirically that color refinement can indeed greatly reduce the cost of solving linear programs, see, e.g., Mladenov et al. [2012], Kersting et al. [2015], Grohe et al. [2014], and variants have been explored in several ways for lifted inference approaches for (relational) graphical models [Bui et al., 2013, Noessner et al., 2013, Mladenov et al., 2014b,a, Apsel et al., 2014, Mladenov and Kersting, 2015]. Very recently, Lu and Boutilier [2015] presented a value-directed compression technique for propositional assignment LPs. They dynamically segment the individuals into blocks using a form of column generation, constructing groups of individuals who can provably be treated identically in the optimal assignment solution.

CHAPTER 10
Conclusions

Real agents need to deal with uncertainty and reason about individuals and relations. They need to learn how the world works before they have encountered all the individuals they need to reason about. If we accept these premises, then we need to get serious about relational (probabilistic) models.

This book has provided an introduction and overview of this new and exciting area that is known as statistical relational AI. It combines principles of statistical and logical AI into a coherent whole. The representations were reviewed starting from a logically parametrized perspective. This turned out to be quite useful as it provides a unifying scheme to talk about relational models. The techniques of statistical relational learning were analyzed starting from a logical (or inductive logic programming) perspective. This turned out to be quite useful for obtaining an appreciation of the differences and similarities among the various frameworks and formalisms. In particular, the distinction between a model- and a proof-theoretic view was used for clarifying the relation among the logical upgrades of graphical models such as MLNs and grammars such as ProbLog. This distinction turned out to be useful for obtaining an appreciation of the differences and similarities among the different learning approaches. The dominating learning setting considers (probabilistic) interpretations as data but there are cases where learning from entailment and from traces are more appropriate. Furthermore, principles of both statistical learning and inductive logic programming are employed for learning the parameters and structure of the relational probabilistic models. Finally, it was illustrated that statistical relational approaches go beyond probabilities and actually cover the whole AI spectrum.

While there have been considerable advances in the last two decades, there are more than enough problems to go around to establish what has come to be called statistical relational AI. Some of the most important ones include the following.

- *Dynamics*: An important class of problems for which graphical models have been very successful concerns dealing with temporal or sequential information, cf. the use of Hidden Markov Models, Conditional Random Fields and Dynamic Bayesian networks in domains such as bio-informatics, speech processing, natural language processing and robotics. While there has been some work on extending probabilistic graphical models for dealing with time, e.g., Manfredotti [2009], Thon et al. [2011], Cattelani et al. [2014], Nitti et al. [2014], we are still far away from a common use of these methods. One of the reasons is that relational states typically evolve over time, making it hard to do inference.

- *Relational probabilistic modeling* is not an easy task and raises several novel issues when it comes to knowledge representation. What assumptions are we making? Why should we choose one representation over another? We may learn a model for some population size(s), and want to apply it to other population sizes. We want to make modeling assumptions explicit and know the consequences of these assumptions. If one model fits some data, it is important to understand why it fits the data better. Kazemi et al. [2014] provided answers to these questions for the case of the logistic regression model but other probabilistic models should be explored since determining which models to use is more than fitting the models to data; we need to understand what we are representing.

- *Decision-theory and planning* are popular in the machine learnig and uncertainty in artificial intelligence communities. While there have been initial attempts, some of which have been sketched in the previous chapter, to obtain decision theoretic statistical relational AI systems, it is still hard to use these systems in practice.

- *Continuous distributions*: This book (as statistical relational AI) has focused on the modeling of discrete probability distributions, but many applications in robotics, vision or speech recognition, require the use of continuous probability distributions. While such distributions are common in contemporary probabilistic programming languages such as Church [Goodman et al., 2008] and BLOG [Milch et al., 2005], they do require approximate inference techniques based on sampling. To illustrate the idea, consider the *distributional clause* extension of ProbLog [Gutmann et al., 2011, Nitti et al., 2014] in which one can write expressions such as $h \sim D \leftarrow b_1, \ldots, b_n$ which states that whenever $(b_1, \ldots, b_n)\theta$ holds the variable $h\theta$ will be distributed according to $D\theta$. One can then also use the values of the random variable $h\theta$ inside clauses as any other term in Prolog. Simpler (typically pairwise) models have already been used to lift Kalman filters [Choi et al., 2010, 2011b, Ahmadi et al., 2011]. Alternatively, it might be useful to explore lifting hybrid models to the relational case [Schiegg et al., 2012, Ravkic et al., 2015].

- *Relational optimization:* Lifting linear programs to the relational case [Kersting et al., 2015] as illustrated in this book already paves the way to relational linear assignment, allocation and many other AI tasks. However, linear programs are only one fragment of the rich world of mathematical programs used throughout AI. Moving beyond linear programs is a growth path for statistical relational AI since it would make mathematical models such as quadratic and semi-definite programs faster to write and easier to understand, reduces development time as well as the level of expertise necessary to build optimization applications, and speed up solvers by exploiting relational properties. Cussens [2015] recently showed the benefits of a relational language for generating and solving mixed integer programs.

- *Approximate lifting and counting* are another set of directions that can have huge potential inside statistical relational AI. The key ideas are based on the observation that when lifting models for performing inference, one could potentially identify approximate symmetries in the model instead of exact ones. While this insight has shown to have significant gains in MAP approximate inference [Bui et al., 2013, Mladenov et al., 2014b,a], the more general problem of identifying these symmetries for exact inference and marginal inference remains an interesting and challenging one. Another type of approximation that can be potentially supported by these statistical models is that of approximate counting. Counting has been identified as the main bottleneck for several probabilistic operations—from sampling to message passing to lifting [Shavlik and Natarajan, 2009, Das et al., 2015, Venugopal et al., 2015]. Obtaining approximate yet reasonable counts can potentially allow for efficient learning and lifting and remains an important but relatively unexplored direction.

- *Applications:* Modern social and technological trends will continue to increase the amount of accessible data, with a significant portion of the resources being interrelated in a complex way and having inherent uncertainty. Making use of this relational data within novel applications remains an important avenue and is likely to yield substantial social and/or business value.

We hope that the book will inspire all of you to join us in exploring the frontiers and the yet unexplored areas of statistical relational AI.

Bibliography

Ahmadi, B., Kersting, K., and Hadiji, F. (2010). Lifted belief propagation: Pairwise marginals and beyond. In *Proc. of the 5th European Workshop on Probabilistic Graphical Models (PGM–10)*. Helsinki, Finland. 90

Ahmadi, B., Kersting, K., Mladenov, M., and Natarajan, S. (2013). Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning*, 92(1): 91–132. DOI: 10.1007/s10994-013-5385-0. 89, 110

Ahmadi, B., Kersting, K., and Sanner, S. (2011). Multi-evidence lifted message passing, with application to PageRank and the Kalman filter. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI–11)*. 90, 136

Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. (1993). *Network flows: theory, algorithms, and applications*. Prentice hall. 130

Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., and Ng, A. (2005). Discriminative learning of Markov random fields for segmentation of 3D scan data. In C. Schmid, S. Soatto, and C. Tomasi (Eds.), *Proc. of the 2005 IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR-05)*. 11

Apsel, U., Kersting, K., and Mladenov, M. (2014). Lifting relational MAP-LPs using cluster signatures. In *Proc. of the 28th AAAI Conference on Artificial Intelligence (AAAI-14)*. 134

Bacchus, F. (1990). *Representing and reasoning with uncertain knowledge*. MIT Press, Cambridge, Massachusetts. 45

Bacchus, F., Halpern, J.Y., and Levesque, H.J. (1999). Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence*, 111(1–2): 171–208. DOI: 10.1016/s0004-3702(99)00031-4. 125

Bach, S.H., Huang, B., London, B., and Getoor, L. (2013). Hinge-loss Markov random fields: Convex inference for structured prediction. In *Proc. of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)*. 34

Baral, C., Gelfond, M., and Rushton, N. (2009). Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming*, 9 (1): 57-144. 74

Baral, C. and Hunsaker, M. (2007). Using the probabilistic logic programming language P-log for causal and counterfactual reasoning and non-naive conditioning. In *Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 243–249. 74

Beame, P., Van den Broeck, G., Gribkoff, E., and Suciu, D. (2015). Symmetric weighted first-order model counting. In *Proc. of the 34th ACM Symposium on Principles of Database Systems (PODS-15)*. DOI: 10.1145/2745754.2745760. 83

Beetz, M., Jain, D., Mösenlechner, L., and Tenorth, M. (2010). Towards performing everyday manipulation activities. *Robotics and Autonomous Systems*, 58. DOI: 10.1016/j.robot.2010.05.007. 10

Beierle, C., Finthammer, M., and Kern-Isberner, G. (2015). Relational probabilistic conditionals and their instantiations under maximum entropy semantics for first-order knowledge bases. *Entropy*, 17(2): 852–865. DOI: 10.3390/e17020852. 34

Bellman, R.E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ. DOI: 10.1126/science.153.3731.34. 124, 125, 126

Bellodi, E. and Riguzzi, F. (2015). Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming*, 15(2): 169–212. DOI: 10.1017/s1471068413000689. 116

Berkholz, C., Bonsma, P., and Grohe, M. (2013). Tight lower and upper bounds for the complexity of canonical colour refinement. In *Proc. of the 21st Annual European Symposium on Algorithms (ESA-13)*. DOI: 10.1007/978-3-642-40450-4_13. 89

Bertsimas, D. and Tsitsiklis, J. (1997). *Introduction to Linear Optimization*. Athena Scientific, 1st ed. 130

Birnbaum, E. and Lozinskii, E.L. (1999). The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, pp. 457–477. 70

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. Springer. 12

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2): 285–297. DOI: 10.1016/s0004-3702(98)00034-4. 117

Boutilier, C., Dean, T., and Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11: 1–94. 10

Boutilier, C., Reiter, R., and Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*. 125

Brachman, R. and Levesque, H. (2004). *Knowledge Representation and Reasoning*. Morgan Kaufmann. 45

Braunstein, A. and Zecchina, R. (2004). Survey propagation as local equilibrium equations. *Journal of Statistical Mechanics*, P06007. DOI: 10.1088/1742-5468/2004/06/p06007. 123

Breese, J.S. (1992). Construction of belief and decision networks. *Computational Intelligence*, 8(4): 624–647. DOI: 10.1111/j.1467-8640.1992.tb00382.x. 33

Bröcheler, M., Mihalkova, L., and Getoor, L. (2010). Probabilistic similarity logic. In *Proc. of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)*. 34

Buchman, D. and Poole, D. (2015). Representing aggregators in relational probabilistic models. In *Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*. 50

Bui, H., Huynh, T., and Riedel, S. (2013). Automorphism groups of graphical models and lifted variational inference. In *Proc. of the 29th Conference on Uncertainty in Artificial Intelligence (UAI-13)*. 90, 134, 137

Bui, H., Huynh, T., and Sontag, D. (2014). Lifted tree-reweighted variational inference. In *Proc. of the 30th Conference on Uncertainty in Artificial Intelligence (UAI-14)*. 90

Buntine, W.L. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2: 159–225. 28, 30, 106, 119

Carbonetto, P., Kisynski, J., de Freitas, N., and Poole, D. (2005). Nonparametric Bayesian logic. In *Proc. 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*. 34, 61

Cattelani, L., Manfredotti, C.E., and Messina, E. (2014). A particle filtering approach for tracking an unknown number of objects with dynamic relations. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 13(1): 3–21. DOI: 10.1007/s10852-012-9213-5. 135

Chavira, M. and Darwiche, A. (2005). Compiling Bayesian networks with local structure. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI-05)*. 51

Cheng, X. and Roth, D. (2013). Relational inference for wikification. In *Proc. of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP-13)*. 130

Chiang, M. and Poole, D. (2011). Reference classes and relational learning. *International Journal of Approximate Reasoning*, 53(3): 326-346. DOI: 10.1016/j.ijar.2011.05.002. 111

Choi, A. and Darwiche, A. (2011). Relax, compensate and then recover. In T. Onada, D. Bekki, and E. McCready (Eds.), *New Frontiers in Artificial Intelligence*, volume 6797 of *Lecture Notes in Computer Science*, pp. 167–180. Springer Berlin/Heidelberg. DOI: 10.1007/978-3-642-25655-4. 90

Choi, J., Amir, E., and Hill, D. (2010). Lifted inference for relational continuous models. In *Proc. of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)*. 87, 136

Choi, J., de Salvo Braz, R., and Bui, H. (2011a). Efficient methods for lifted inference with aggregate factors. In *Proc. of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*. 87

Choi, J., Guzman-Rivera, A., and Amir, E. (2011b). Lifted relational Kalman filtering. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*. 87, 136

Chu, W., Sindhwani, V., Ghahramani, Z., and Keerthi, S. (2005). Relational learning with Gaussian processes. In *Advances in Neural Information Processing Systems 18 (NIPS-05)*. 34

Clark, K.L. (1978). Negation as failure. In H. Gallaire and J. Minker (Eds.), *Logic and Databases*, pp. 293–322. Plenum Press, New York. DOI: 10.1007/978-1-4684-3384-5. 53

Clarke, J. and Lapata, M. (2008). Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31: 399–429. 130

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3): 245–271. 13

Cussens, J. (2015). First-order integer programming for map problems. In *Working Notes of the 5th International Workshop on Statistical Relational AI (StarAI@UAI-15)*. ArXiv:1507.02912. 136

Darwiche, A. (2002). A logical approach to factoring belief networks. In *Proc. of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, Morgan Kaufmann. 51

Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, 126(1-2): 5–41. DOI: 10.1016/s0004-3702(00)00069-2. 66

Darwiche, A. (2004). New advances in compiling CNF to decomposable negation normal form. In *Proc. of the 16th Eureopean Conference on Artificial Intelligence (ECAI-04)*, IOS Press. 80

Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press. DOI: 10.1017/cbo9780511811357. 18, 80

Darwiche, A. and Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17(1): 229–264. DOI: 10.1613/jair.989. 80, 111

Das, M., Wu, Y., Khot, T., Kersting, K., and Natarajan, S. (2015). Graph-based approximate counting for relational probabilistic models. In *Working Notes of the 5th International Workshop on Statistical Relational AI (StarAI@UAI-15)*. 137

Davis, J. and Domingos, P.M. (2009). Deep transfer via second-order Markov logic. In *Proc. of the 26th International Conference on Machine Learning (ICML-09)*. DOI: 10.1145/1553374.1553402. 11

de Finetti, B. (1974). *Theory of Probability*. Wiley, New York. 28

De Maeyer, D., Renkens, J., Cloots, L., De Raedt, L., and Marchal, K. (2013). Phenetic: network-based interpretation of unstructured gene lists in E. coli. *Molecular BioSystems*, 9(7): 1594–1603. DOI: 10.1039/c3mb25551d. 12

De Raedt, L. (1992). *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press. 117

De Raedt, L. and Dehaspe, L. (1997). Clausal Discovery. *Machine Learning*, 26(2-3): 99–146. 100

De Raedt, L. and Kersting, K. (2004). Probabilistic Inductive Logic Programming. In S. Ben-David, J. Case, and A. Maruoka (Eds.), *Proc. of the 15th International Conference on Algorithmic Learning Theory (ALT-04)*, volume 3244 of *Lecture Notes in Computer Science*, pp. 19–36. Springer. 114

De Raedt, L. and Kersting, K. (2010). Statistical relational learning. In C. Sammut and G. Webb (Eds.), *Encyclopedia of Machine Learning*, pp. 916–924. Springer, Heidelberg. DOI: 10.1007/978-0-387-30164-8. 114

De Raedt, L., Kimmig, A., and Toivonen, H. (2007). ProbLog: A probabilistic Prolog and its application in link discovery. In *Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*. 34, 38, 42, 52

De Raedt, L. (2008). *Logical and Relational Learning*. Springer. DOI: 10.1007/978-3-540-68856-3. 13, 96, 102

De Raedt, L. (2010). Logic of generality. In *Encyclopedia of Machine Learning*, pp. 624–631. Springer. DOI: 10.1007/978-0-387-30164-8_489. 97, 100, 102

De Raedt, L., Dries, A., Thon, I., Van den Broeck, G., and Verbeke, M. (2015). Inducing probabilistic relational rules from probabilistic examples. In *Proc. of 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*. 117

De Raedt, L. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100(1): 5–47. DOI: 10.1007/s10994-015-5494-z. 42, 50

de Salvo Braz, R., Natarajan, S., Bui, H., Shavlik, J., and Russell, S. (2009). Anytime lifted belief propagation. In *Working notes of the Statistical Relational Learning Workshop (SRL-09)*. 88

de Salvo Braz, R., Amir, E., and Roth, D. (2005). Lifted first-order probabilistic inference. In *In Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*. 82

de Salvo Braz, R., Amir, E., and Roth, D. (2007). Lifted first-order probabilistic inference. In L. Getoor and B. Taskar (Eds.), *Introduction to Statistical Relational Learning*. MIT Press. 81, 82

Dean, T. and Kanazawa, K. (1988). Probabilistic Temporal Reasoning. In T. Mitchell and R. Smith (Eds.), *Proc. of the 7th National Conference on Artificial Intelligence (AAAI-88)*, AAAI Press/The MIT Press. 110

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39: 1–39. 95, 109

Dhurandhar, A. and Dobra, A. (2012). Distribution-free bounds for relational classification. *Knowledge and Information Systems*, 31(1): 55–78. DOI: 10.1007/s10115-011-0406-4. 112

Di Mauro, N., Bellodi, E., and Riguzzi, F. (2015). Bandit-based Monte-Carlo structure learning of probabilistic logic programs. *Machine Learning*, 100(1): 127–156. DOI: 10.1007/s10994-015-5510-3. 116

Domingos, P. and Lowd, D. (2009). Markov logic: An interface layer for artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1): 1–155. DOI: 10.2200/s00206ed1v01y200907aim007. 109, 112

Dong, X.L., Gabrilovich, E., Heitz, G., Horn, W., Murphy, K., Sun, S., and Zhang, W. (2014). From data fusion to knowledge fusion. *Proc. of the VLDB Endowment* 7(10): 881–892. DOI: 10.14778/2732951.2732962. 8

Driessens, K. and Dzeroski, S. (2002). Integrating experimentation and guidance in relational reinforcement learning. In *Proc. of the 19th International Conference on Machine Learning (ICML-02)*. 128

Driessens, K. and Dzeroski, S. (2005). Combining model-based and instance-based learning for first-order regression. In *Proc. of the 22nd International Conference on Machine Learning (ICML-05)*. DOI: 10.1145/1102351.1102376. 128

Driessens, K. and Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. In *Proc. of the 20th International Conference on Machine Learning (ICML-03)*. 128

Driessens, K., Ramon, J., and Gärtner, T. (2006). Graph kernels and Gaussian processes for relational reinforcement learning. *Machine Learning*, 64(1-3): 91-119. DOI: 10.1007/s10994-006-8258-y. 128

Dzeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43: 7–52. DOI: 10.1007/10.1023/A:1007694015589. 128

Fellegi, I. and Sunter, A. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328): 1183–1280. DOI: 10.2307/2286061. 58, 59

Fern, A., Yoon, S., and Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25: 75–118. 128

Fern, A., Yoon, S., and Givan, R. (2003). Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing 16 (NIPS-03)*. 127

Fierens, D., Van den Broeck, G., Renkens, J., Sht. Shterionov, D., Gutmann, B., Thon, I., Janssens, G., and De Raedt, L. (2015). Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3): 358–401. DOI: 10.1017/s1471068414000076. 78, 79, 80, 107, 111

Fikes, R.E. and Nilsson, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4): 189–208. DOI: 10.1016/0004-3702(71)90010-5. 125

Flach, P. (1994). *Simply Logical: Intelligent Reasoning by Example*. John Wiley. DOI: 10.5860/choice.32-2788. 25

Foulds, J.R., Kumar, S.H., and Getoor, L. (2015). Latent topic networks: A versatile probabilistic programming framework for topic models. In *Proc. of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 777–786. 12

Frey, B., Kschischang, F., Loeliger, H.A., and Wiberg, N. (1997). Factor graphs and algorithms. In *Proc. of the 35th Allerton Conference on Communication, Control, and Computing*, pp. 666–680. Champaign-Urbana, IL. 51

Friedman, N. (1998). The Bayesian Structural EM Algorithm. In G. Cooper and S. Moral (Eds.), *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*. Morgan Kaufmann. 114

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning Probabilistic Relational Models. In T. Dean (Ed.), *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*. 13, 34, 54, 108, 109, 115

Geffner, H. (2014). Artificial intelligence: From programs to solvers. *AI Communications*, 27(1): 45–51. 129

Gelman, A., Carlin, J.B., Stern, H.S., and Rubin, D.B. (2004). *Bayesian Data Analysis*, 2nd ed., Chapman and Hall/CRC. 108

Getoor, L. (2001). *Learning Statistical Models from Relational Data*. Ph.D. thesis, Stanford University, Stanford, California. DOI: 10.1145/1989323.1989451. 107, 109, 115

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (2001a). Learning Probabilistic Relational Models. In S. Džeroski and N. Lavrač (Eds.), *Relational Data Mining*. Springer. DOI: 10.1007/978-3-662-04599-2. 109, 112, 115, 119

Getoor, L., Friedman, N., Koller, D., and Taskar, B. (2001b). Learning Probabilistic Models of Relational Structure. In C. Brodley and A. Pohoreckyj Danyluk (Eds.), *Proc. of the 18th International Conference on Machine Learning (ICML-01)*. Morgan Kaufmann. 115

Getoor, L., Koller, D., Taskar, B., and Friedman, N. (2000). Learning probabilistic relational models with structural uncertainty. In L. Getoor and D. Jensen (Eds.), *Proc. of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*. 115

Getoor, L., Rhee, J., Koller, D., and Small, P. (2004). Understanding tuberculosis epidemiology using probabilistic relational models. *Artificial Intelligence in Medicine*, 30: 233–256. DOI: 10.1016/j.artmed.2003.11.003. 10

Getoor, L. and Taskar, B. (Eds.) (2007). *Introduction to Statistical Relational Learning*. The MIT Press. 13

Getoor, L., Taskar, B., and Koller, D. (2001c). Using probabilistic models for selectivity estimation. In *Proc. of the 9th ACM SIGMOD International Conference on Management of Data (SIGMOD-01)*. ACM Press. 10, 13

Globerson, A. and Jaakkola, T. (2007). Fixing max-product: Convergent message passing algorithms for map LP-relaxations. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*. 90

Gogate, V. and Domingos, P. (2011). Probabilistic theorem proving. In *Proc. of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*. 53, 82, 87

Gogate, V. and Domingos, P. (2010). Exploiting logical structure in lifted probabilistic inference. In *Working Notes of the 1st Workshop on Statististical Relational AI (STARAI@AAAI-10)*. 82

Goldman, R.P. and Charniak, E. (1990). Dynamic construction of belief networks. In *Proc. of the 6th Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pp. 90–97. 33

Goodman, N., Mansinghka, V., Roy, D.M., Bonawitz, K., and Tenenbaum, J. (2008). Church: a language for generative models. In *Proc. of the 24th Conference on Uncertainty in Artificial Intelligence (UAI-08)*. 13, 34, 119, 136

Gretton, C. and Thiebaux, S. (2004). Exploiting first-order regression in inductive policy selection. In *Proc. of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-04)*. 127

Gribkoff, E., Van den Broeck, G., and Suciu, D. (2014). Understanding the complexity of lifted inference and asymmetric weighted model counting. In *Proc. of the 30th Conference on Uncertainty in Artificial Intelligence (UAI-14)*. 83

Grohe, M., Kersting, K., Mladenov, M., and Selman, E. (2014). Dimension reduction via colour refinement. In *Proc. of the 22th European Symposium on Algorithms (ESA-14)*, pp. 505–516. DOI: 10.1007/978-3-662-44777-2_42. 134

Guns, T., Nijssen, S., and De Raedt, L. (2011). Itemset mining: A constraint programming perspective. *Artificial Intelligence*, 175(12-13): 1951–1983. DOI: 10.1016/j.artint.2011.05.002. 129

Gutmann, B. and Kersting, K. (2006). TildeCRF: Conditional random fields for logical sequences. In *Proc. of the 17th European Conference on Machine Learning (ECML–2006)*, volume 4212 of *Lecture Notes in Computer Science*, pp. 174–185. Springer. 117

Gutmann, B., Kimmig, A., De Raedt, L., and Kersting, K. (2008). Parameter learning in probabilistic databases: A least squares approach. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008)*, volume 5211 of *Lecture Notes In Computer Science*, pp. 473–488. Springer. 111

Gutmann, B., Thon, I., Kimmig, A., Bruynooghe, M., and De Raedt, L. (2011). The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11(4-5): 663–680. DOI: 10.1017/s1471068411000238. 136

Hadiji, F., Ahmadi, B., and Kersting, K. (2011). Efficient sequential clamping for lifted message passing. In *Proc. of the 34th German Conference on Artificial Intelligence (KI–11)*, pp. 122–133. DOI: 10.1007/978-3-642-24455-1_11. 90

Hadiji, F., Kersting, K., and Ahmadi, B. (2010). Lifted message passing for satisfiability. In *Working Notes of the Workshop on Statistical Relational AI (StarAI@AAAI-10)*. 90, 124

Hadiji, F., Mladenov, M., Bauckhage, C., and Kersting, K. (2015). Computer science on the move: Inferring migration regularities from the web via compressed label propagation. In *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*. 12, 90

Halpern, J.Y. (1990). An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3): 311–350. DOI: 10.1016/0004-3702(90)90019-v. 13, 17, 45

Halpern, J.Y. (2003). *Reasoning about Uncertainty*. MIT Press, Cambridge, MA. 17, 18, 58

Heckerman, D. (1995). A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research. (Revised November 1996). DOI: 10.1007/978-3-540-85066-3_3. 109, 113

Heckerman, D. (1990). Probabilistic Similarity Networks. Ph.D. thesis, Stanford Universityersity. DOI: 10.1002/net.3230200508. 59

Heckerman, D., Meek, C., and Koller, D. (2004). Probabilistic models for relational data. Technical Report MSR-TR-2004-30, Microsoft Research. 34, 48

Hescott, B.J. and Khardon, R. (2014). The complexity of reasoning with FODD and GFODD. In *Proc. of the 28th Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*. DOI: 10.1016/j.artint.2015.08.005. 128

Hölldobler, S. and Skvortsova, O. (2004). A logic-based approach to dynamic programming. In *Proc. of the AAAI-04 Workshop on Learning and Planning in MDPs*. 127

Horsch, M. and Poole, D. (1990). A dynamic approach to probabilistic inference using Bayesian networks. In *Proc. of the 6th Conference on Uncertainty in Artificial Intelligence (UAI-90)*, pp. 155–161. Boston. 33, 54

Jaeger, M. (2007). Parameter learning for relational Bayesian networks. In *Proc. of the 24th International Conference on Machine Learning (ICML-07)*. DOI: 10.1145/1273496.1273543. 57, 108, 110

Jaeger, M. (2000). On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117: 297–308. DOI: 10.1016/s0004-3702(99)00109-5. 83

Jaeger, M. and Van Den Broeck, G. (2012). Liftability of probabilistic inference: Upper and lower bounds. In *Working Notes of the Workshop on Statistical Relational AI (StarAI@AAAI-12)*. 83, 112

Jaeger, M. (1997). Relational Bayesian Networks. In K. Laskey and H. Prade (Eds.), *Proc. of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*. Morgan Kaufmann. 13

Jaimovich, A., Meshi, O., and Friedman, N. (2007). Template-based inference in symmetric relational Markov random fields. In *Proc. of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*. 88

Janhunen, T. (2004). Representing normal programs with clauses. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*. IOS Press. 73, 79

Janowicz, K., van Harmelen, F., Hendler, J.A., and Hitzler, P. (2015). Why the data train needs semantic rails. *AI Magazine*, 36(1): 5–14. 62

Jaynes, E.T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press. DOI: 10.1017/cbo9780511790423. 45

Jensen, D. and Neville, J. (2002). Linkage and autocorrelation cause feature selection bias in relational learning. In *Proc. of the 19th International Conference on Machine Learning (ICML-02)*. 112

Jensen, D., Neville, J., and Hay, M. (2003). Avoiding bias when aggregating relational data with degree disparity. In *Proc. of the 20th International Conference on Machine Learning (ICML-03)*, pp. 274–281. 112

Jensen, F.V., Lauritzen, S.L., and Olesen, K.G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4: 269–282. 66

Jernite, Y., Rush, A.M., and Sontag, D. (2015). A fast variational approach for learning Markov random field language models. In *Proc. of the 32nd International Conference on Machine Learning (ICML-15)*. 8, 9

Jha, A., Gogate, V., Meliou, A., and Suciu, D. (2010). Lifted inference from the other side: The tractable features. In *Advances in Neural Information Processing Systems 23 (NIPS-10)*. 82

Jordan, M.I. (2010). Bayesian nonparametric learning: Expressive priors for intelligent systems. In R. Dechter, H. Geffner, and J.Y. Halpern (Eds.), *Heuristics, Probability and Causality: A Tribute to Judea Pearl*, pp. 167–186. College Publications. 28, 30, 106, 119

Joshi, S., Kersting, K., and Khardon, R. (2010). Self-taught decision theoretic planning with first order decision diagrams. In *Proc. of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*. 127

Kallenberg, O. (2005). *Probabilistic Symmetries and Invariance Principles*. Springer. DOI: 10.1007/0-387-28861-9. 28

Kameya, Y. and Sato, T. (2000). Efficient EM learning with tabulation for parameterized logic programs. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, and P. Stuckey (Eds.), *Proc. of the 1st International Conference on Computational Logic (CL-00)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pp. 269–294. Springer-Verlag. 111

Kameya, Y., Ueda, N., and Sato, T. (1999). A graphical method for parameter learning of symbolic-statistical models. In *Proc. of the 2nd International Conference on Discovery Science (DS-99)*, volume 1721 of *Lectures Notes in Artificial Intelligence*, pp. 254–276. Springer-Verlag, Kanagawa, Japan. DOI: 10.1007/3-540-46846-3_24. 111

Karabaev, E. and Skvortsova, O. (2005). A heuristic search algorithm for solving first-order MDPs. In *Proc. of the 25th Conference on Uncertainty in Artificial Intelligence (UAI-2005)*Edinburgh, Scotland. 127

Kazemi, S.M., Buchman, D., Kersting, K., Natarajan, S., and Poole, D. (2014). Relational logistic regression. In *Proc. of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR-14)*. 136

Kemp, C., Tenenbaum, J., Griffiths, T., Yamada, T., and Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *Proc. of the 21st AAAI Conference on Artificial Intelligence (AAAI-06)*. 34

Kern-Isberner, G. and Thimm, M. (2010). Novel semantical approaches to relational probabilistic conditionals. In *Proc. of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR-10)*. 34

Kersting, K., Ahmadi, B., and Natarajan, S. (2009). Counting belief propagation. In J.B.A. Ng (Ed.), *Proc. of the 25th Conference on Uncertainty in Artificial Intelligence (UAI–09)*. 88, 89

Kersting, K. and De Raedt, L. (2001). Adaptive Bayesian Logic Programs. In C. Rouveirol and M. Sebag (Eds.), *Proc. of the 11th Conference on Inductive Logic Programming (ILP-01)*, volume 2157 of *Lecture Notes in Computer Science*, pp. 104–117. Springer. 13, 109, 110

Kersting, K. and De Raedt, L. (2007). Bayesian logic programming: Theory and tool. In L. Getoor and B. Taskar (Eds.), *An Introduction to Statistical Relational Learning*. MIT Press. 34, 57, 107, 109, 115

Kersting, K. and De Raedt, L. (2008). Probabilistic inductive logic programming. In L. De Raedt et al. (Ed.), *Probabilistic Inductive Logic Programming: Theory and Applications*, volume 4911 of *Lecture Notes in Artificial Intelligence*, pp. 1–28. Springer. DOI: 10.1007/978-3-540-78652-8_1. 114

Kersting, K., De Raedt, L., and Raiko, T. (2006). Logical Hidden Markov Models. *Journal of Artificial Intelligence Research*, 25: 425–456. 11

Kersting, K. and Driessens, K. (2008). Non–parametric policy gradients: A unified treatment of propositional and relational domains. In S.R.A. McCallum (Ed.), *Proc. of the 25th International Conference on Machine Learning (ICML-08)*. Helsinki, Finland. DOI: 10.1145/1390156. 108, 117, 128

Kersting, K., Massaoudi, Y.E., Ahmadi, B., and Hadiji, F. (2010). Informed lifting for message–passing. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI–10)*. 88

Kersting, K., Mladenov, M., and Tokmakov, P. (2015). Relational linear programs. *Artificial Intelligence*. In press. DOI: 10.1016/j.artint.2015.06.009. 12, 130, 131, 133, 134, 136

Kersting, K. and Xu, Z. (2009). Learning preferences with hidden common cause relations. In *Proc. of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 09)*, volume 5781 of *Lecture Notes in Computer Science*, pp. 676–691. Springer. DOI: 10.1007/978-3-642-04180-8_61. 11

Kersting, K., van Otterlo, M., and De Raedt, L. (2004). Bellman goes relational. In *Proc. of the 21st International Conference on Machine Learning (ICML-04)*. ACM Press. DOI: 10.1145/1015330.1015401. 125, 127

Khosravi, H., Schulte, O., Hu, J., and Gao, T. (2012). Learning compact Markov logic networks with decision trees. *Machine Learning*, 89(3): 257–277. DOI: 10.1007/s10994-012-5307-6. 119

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. (2011). Learning Markov logic networks via functional gradient boosting. In *Proc. of the 11th IEEE Conference on Data Mining (ICDM-11)*. DOI: 10.1109/icdm.2011.87. 116, 119

Khot, T., Natarajan, S., Kersting, K., and Shavlik, J. (2015a). Gradient-based boosting for statistical relational learning: The Markov logic network and missing data cases. *Machine Learning*, 100(1). DOI: 10.1007/s10994-015-5481-4. 119

Khot, T., Balasubramanian, N., Gribkoff, E., Sabharwal, A., Clark, P., and Etzioni, O. (2015b). Exploring Markov logic networks for question answering. In *Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP-15)*. DOI: 10.18653/v1/d15-1080. 12

Kiddon, C. and Domingos, P. (2011). Coarse-to-fine inference and learning for first-order probabilistic models. In *Proc. of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*. 11

Kisyński, J. and Poole, D. (2009). Lifted aggregation in directed first-order probabilistic models. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*. 87

Kok, S. and Domingos, P. (2005). Learning the structure of Markov Logic Networks. In *Proc. of the 22nd International Conference on Machine Learning (ICML-05)*. DOI: 10.1145/1102351. 115

Kok, S. and Domingos, P. (2007). Learning the structure of Markov logic networks. In *Proc. of the 24th International Conference on Machine Learning (ICML-07)*. DOI: 10.1145/1102351.1102407. 112

Kok, S. and Domingos, P. (2009). Learning Markov Logic Network Structure via Hypergraph Lifting. In *Proc. of the 26th International Conference on Machine Learning (ICML-09)*. DOI: 10.1145/1553374.1553440. 116

Kok, S. and Domingos, P. (2010). Using structural motifs for learning Markov logic networks. In *Working Notes of the Workshop on Statistical Relational Artificial Intelligence (StarAI@AAAI-10)*. 116

Koller, D. and Pfeffer, A. (1997). Learning probabilities for noisy first-order rules. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*. 109, 110

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. 12, 18, 29, 96, 109

Kordjamshidi, P., Roth, D., and Wu, H. (2015). Saul: Towards declarative learning based programming. In *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*. 130

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8): 30–37. DOI: 10.1109/mc.2009.263. 112

Kowalski, R. (2014). *Logic for Problem Solving, Revisited*. Books On Demand. 23, 45

Kschischang, F.R., Frey, B.J., and Loeliger, H.A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2): 498-519. DOI: 10.1109/18.910572. 123

Kyburg, H. (1983). The reference class. *Philosophy of Science*, 50: 374–397. DOI: 10.1086/289125. 105

Lang, T. and Toussaint, M. (2009). Relevance grounding for planning in relational domains. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD-09)*. volume 5781 of *Lecture Notes in Computer Science*, pp. 736–751. Springer. DOI: 10.1007/978-3-642-04180-8_65. 127

Lang, T. and Toussaint, M. (2010). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39: 1–49. 128

Lang, T., Toussaint, M., and Kersting, K. (2010). Exploration in relational worlds. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD-10)*. volume 6322 of *Lecture Notes in Computer Science*, pp. 178–194. Springer. 129

Lang, T., Toussaint, M., and Kersting, K. (2012). Exploration in relational domains for model-based reinforcement learning. *Journal of Machine Learning Research*, 13: 3691–3734. 12, 128

Laskey, K.B. (2008). MEBN: A language for first-order Bayesian knowledge bases. *Artificial Intelligence*, 172(2-3): 140–178. DOI: 10.1016/j.artint.2007.09.006. 34, 48

Lauritzen, S.L. and Spiegelhalter, D.J. (1988). 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. 66, 72

Lauritzen, S. (1996). *Graphical Models*. Oxford University Press. 18

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, NY. 13

Limketkai, B., Liao, L., and Fox, D. (2005). Relational Object Maps for Mobile Robots. In F. Giunchiglia and L.P. Kaelbling (Eds.), *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*. AAAI Press. 11

Lloyd, J.W. (1989). *Foundations of Logic Programming*, 2nd ed., Springer, Berlin. DOI: 10.1007/978-3-642-96826-6. 25

Lu, T. and Boutilier, C. (2015). Value-directed compression of large-scale assignment problems. In *Proc. of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*. 134

Mallory, E.K., Zhang, C., Ré, C., and Altman, R.B. (2016). Large-scale extraction of gene interactions from full-text literature using deepdive. *Bioinformatics*, 32(1): 106–113. DOI: 10.1093/bioinformatics/btv476. 12

Manfredotti, C.E. (2009). Modeling and inference with relational dynamic Bayesian networks. In *Proc. of the 22nd Canadian Conference on Artificial Intelligence*. DOI: 10.1007/978-3-642-01818-3_44. 135

Marlin, B.M., Zemel, R.S., Roweis, S.T., and Slaney., M. (2011). Recommender systems, missing data and statistical model estimation. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*. 95

Martins, A., Smith, N., and Xing, E. (2009). Concise integer linear programming formulations for dependency parsing. In *Proc. of the 47th Annual Meeting of the Association for Computational Linguistics (ACL-09)*, pp. 342–350. DOI: 10.3115/1687878.1687928. 130

McCarthy, J. and Hayes, P.J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In M. Meltzer and D. Michie (Eds.), *Machine Intelligence 4*, pp. 463–502. Edinburgh University Press. 45

McGovern, A., Friedland, L., Hay, M., Gallagher, B., Fast, A., Neville, J., and Jensen, D. (2003). Exploiting relational structure to understand publication patterns in high-energy physics. *SIGKDD Explorations*, 5(2): 165–173. DOI: 10.1145/980972.980999. 10

McLachlan, G. and Krishnan, T. (1997). *The EM Algorithm and Extensions*. Wiley, New York. DOI: 10.1002/9780470191613. 95, 109

Meza-Ruíz, I.V. and Riedel, S. (2009). Jointly identifying predicates, arguments and senses using Markov logic. In *Proc. of the Conference of the North American Chapter of the Association of Computational Linguistics (NACL-09)*. DOI: 10.3115/1620754.1620777. 11

Mézard, M., Parisi, G., and Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, 297: 812–815. DOI: 10.1126/science.1073287. 123

Mihalkova, L. and Mooney, R. (2007). Bottom-up learning of Markov Logic Network structure. In *Proc. of the 24th International Conference on Machine Learning (ICML-07)*. DOI: 10.1145/1273496.1273575. 112, 116

Mihalkova, L. and Richardson, M. (2010). Speeding up inference in statistical relational learning by clustering similar query literals. In *Proc. of the 19th International Conference on Inductive Logic Programming (ILP-09)*. volume 5989 of *Lecture Notes in Computer Science*, pp. 178–194. Springer. DOI: 10.1007/978-3-642-13840-9_11. 88

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., and Kolobov, A. (2005). BLOG: Probabilistic models with unknown objects. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*. Edinburgh. 13, 58, 60, 61, 136

Milch, B. and Russell, S. (2006). General-purpose MCMC inference over relational structures. In *Proc. of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI-06)*. 87

Milch, B., Zettlemoyer, L.S., Kersting, K., Haimes, M., and Kaelbling, L.P. (2008). Lifted probabilistic inference with counting formulas. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*. 52, 81, 82

Mitchell, T.M. (1997). *Machine Learning*. McGraw-Hill. 96, 98

Mittal, H., Goyal, P., Gogate, V.G., and Singla, P. (2014). New rules for domain independent lifted MAP inference. In *Advances in Neural Information Processing Systems 27 (NIPS-14)*. 82

Mladenov, M., Ahmadi, B., and Kersting, K. (2012). Lifted linear programming. In *Proc. of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, pp. 788–797. Volume 22 of the JMLR Proceedings. 90, 134

Mladenov, M., Globerson, A., and Kersting, K. (2014a). Lifted message passing as reparametrization of graphical models. In *Proc. of the 30th Conference on Uncertainty in Artificial Intelligence (UAI-14)*. 90, 134, 137

Mladenov, M. and Kersting, K. (2015). Equitable partitions of concave free energies. In *Proc. of the 31st Conference on Uncertainty in Artificial Intelligence (UAI-15)*. 90, 134

Mladenov, M., Kersting, K., and Globerson, A. (2014b). Efficient lifting of MAP LP relaxations using k-locality. In *Proc. of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS-14)*. pp. 623–632. Volume 33 of the JMLR Proceedings. 90, 134, 137

Mohan, K. and Pearl, J. (2014). Graphical models for recovering probabilistic and causal queries from missing data. In M. Welling, Z. Ghahramani, C. Cortes, and N. Lawrence (Eds.), *Advances in Neural Information Processing 27 (NIPS-14)*. 95

Moldovan, B. and De Raedt, L. (2014). Learning relational affordance models for two-arm robots. In *Proc. of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. DOI: 10.1109/iros.2014.6942964. 12

Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in Inductive Logic Programming*, pp. 254–264. IOS Press. 34

Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proc. of the 1st International Workshop on Algorithmic Learning Theory (ALT-90)*, pp. 368–381. Springer/Ohmsma, Tokyo, Japan. 99

Muggleton, S. (2002). Learning Structure and Parameters of Stochastic Logic Programs. In S. Matwin and C. Sammut (Eds.), *Proc. of the 12th International Conference on Inductive Logic Prgramming (ILP-02)*, volume 2583 of *Lectures Notes in Computer Science*. Springer. 117

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming*, 19(20): 629–679. DOI: 10.1016/0743-1066(94)90035-3. 13, 96

Murphy, K.P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT press. 12

Murphy, K., Weiss, Y., and Jordan, M. (1999). Loopy Belief Propagation for Approximate Inference: An Empirical Study. In *Proc. of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*. 68

Natarajan, S., Joshi, S., Saha, B., Edwards, A., E. Moody, T.K., Kersting, K., Whitlow, C., and Maldjian, J. (2012a). A machine learning pipeline for three-way classification of alzheimer patients from structural magnetic resonance images of the brain. In *Proc. of the IEEE Conference on Machine Learning and Applications (ICMLA-12)*. DOI: 10.1109/icmla.2012.42. 12

Natarajan, S., Kersting, K., Ip, E., Jacobs, D., and Carr, J. (2013). Early prediction of coronary artery calcification levels using machine learning. In *Proc. of the 25th Innovative Applications of Artificial Intelligence Conference (IAAI-13)*. 6

Natarajan, S., Khot, T., Kersting, K., and Shavlik, J. (2015). *Boosted Statistical Relational Learners: From Benchmarks to Data-Driven Medicine*. Springer Briefs in Computer Science. 117

Natarajan, S., Tadepalli, P., Dietterich, T.G., and Fern, A. (2009). Learning first-order probabilistic models with combining rules. *Annals of Mathematics and Artificial Intelligence*, 54(1): 223-256. DOI: 10.1145/1102351.1102428. 57, 108, 109, 110

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., and Shavlik, J.W. (2012b). Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1): 25–56. DOI: 10.1007/s10994-011-5244-9. 117, 119

Natarajan, S., Khot, T., Lowd, D., Tadepalli, P., Kersting, K., Shavlik, J., and Iais, F. (2010). Exploiting causal independence in Markov logic networks: Combining undirected and directed models. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD-10)*. volume 6322 of *Lecture Notes in Computer Science*, pp. 434–450. Springer. DOI: 10.1007/978-3-642-15883-4_28. 57

Nath, A. and Domingos, P. (2010). Efficient lifting for online probabilistic inference. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. 90

Neumann, M., Kersting, K., and Ahmadi, B. (2011). Markov logic sets: Towards lifted information retrieval using pagerank and label propagation. In *Proc. of the 25th AAAI Conference on Artificial Intelligence (AAAI–11)*. 90

Neville, J. and Jensen, D. (2004). Dependency Networks for Relational Data. In R. Rastogi, K. Morik, M. Bramer, and X. Wu (Eds.), *Proc. of the 4th IEEE International Conference on Data Mining (ICDM-04)*. 13

Neville, J., Simsek, Ö., Jensen, D., Komoroske, J., Palmer, K., and Goldberg, H. (2005). Using relational knowledge discovery to prevent securities fraud. In *Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press. DOI: 10.1145/1081870.1081922. 10, 54

Neville, J. and Jensen, D. (2007). Relational dependency networks. *Journal of Machine Learning Research*, 8: 653–692. 49, 117

Niepert, M. (2012). Markov chains on orbits of permutation groups. In *Proc. of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)*. 87

Niepert, M., Meilicke, C., and Stuckenschmidt, H. (2010). A probabilistic-logical framework for ontology matching. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. 11

Nilsson, N. (1986). *Principles of Artificial Intelligence*. Springer-Verlag, New York. Reprint, Originally published: Tioga Publishing Co. , 1980. DOI: 10.1007/978-3-662-09438-9. 13

Nitti, D., Belle, V., and De Raedt, L. (2015). Planning in discrete and continuous Markov decision processes by probabilistic programming. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD-15)*. volume 9284 of *Lecture Notes in Computer Science*. pp. 327–342. Springer. DOI: 10.1007/978-3-319-23525-7_20. 128

Nitti, D., De Laet, T., and De Raedt, L. (2014). Relational object tracking and learning. In *Proc. of the IEEE International Conference on Robotics and Automation, (ICRA-14)*. DOI: 10.1109/icra.2014.6906966. 12, 135, 136

Niu, F., Zhang, C., Ré, C., and Shavlik, J.W. (2012). Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *International Journal on Semantic Web and Information Systems*, 8(3): 42–73. DOI: 10.4018/jswis.2012070103. 7

Noessner, J., Niepert, M., and Stuckenschmidt, H. (2013). Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *Proc. of the 27th AAAI Conference on Artificial Intelligence (AAAI-13)*. 134

Odom, P., Bangera, V., Khot, T., Page, D., and Natarajan, S. (2015). Extracting adverse drug events from text using human advice. In *Proc. of the Conference on Artificial Intelligence in Medicine*. Volume 9105 of *Lecture Notes in Computer Science*. pp 195-204. Springer. DOI: 10.1007/978-3-319-19551-3_26. 12

Pasula, H., Marthi, B., Milch, B., Russell, S., and Shpitser, I. (2003). Identity uncertainty and citation matching. In *Advances in Neural Information Processing Systems 15 (NIPS-03)*. 58, 60

Pasula, H., Zettlemoyer, L., and Pack Kaelbling, L. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29: 309–352. 125, 128

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA. 18, 19, 20, 34, 43, 45, 48, 68

Pearl, J. (2009). *Causality: Models, Reasoning and Inference*, 2nd ed., Cambridge University Press. DOI: 10.1017/cbo9780511803161. 47, 106

Perlich, C. and Provost, F. (2006). Distribution-based aggregation for relational learning with identifier attributes. In *Machine Learning*, 62, pp. 65–105. DOI: 10.1007/s10994-006-6064-1. 54

Pfeffer, A. (2001). IBAL: A Probabilistic Rational Programming Language. In *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*. Morgan Kaufmann. 13

Pfeffer, A. (2007). The design and implementation of IBAL: A general-purpose probabilistic language. In L. Getoor and B. Taskar (Eds.), *Statistical Relational Learning*. MIT Press. 34, 119

Plotkin, G.D. (1970). A note on inductive generalization. In *Machine Intelligence*, 5, pp. 153–163. Edinburgh University Press. 102

Poole, D. (1991). Representing diagnostic knowledge for probabilistic Horn abduction. In *Proc. 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*. 33, 37, 40, 52

Poole, D. (1993a). Logic programming, abduction and probability: A top-down anytime algorithm for computing prior and posterior probabilities. *New Generation Computing*, 11(3–4): 377–400. DOI: 10.1007/bf03037184. 48, 77, 79, 88

Poole, D. (1993b). Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64: 81–129. DOI: 10.1016/0004-3702(93)90061-f. 13, 33, 34, 37, 40, 52, 53

Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94: 7–56. Special issue on economic principles of multi-agent systems. DOI: 10.1016/s0004-3702(97)00027-1. 33, 37, 42, 48, 53, 125

Poole, D. (2003). First-order probabilistic inference. In *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 985–991. 13, 28, 52, 81

Poole, D., Bacchus, F., and Kisyński, J. (2011). Towards completely lifted search-based probabilistic inference. *CoRR*, abs/1107.4035. 82, 86

Poole, D. (1998). Decision theory, the situation calculus and conditional plans. *Electronic Transactions on Artificial Intelligence*, 2(1–2). 125

Poole, D. (2007). Logical generative models for probabilistic reasoning about existence, roles and identity. In *Proc. of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*. 58, 60, 61

Poole, D. (2008). The independent choice logic and beyond. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton (Eds.), *Probabilistic Inductive Logic Programming: Theory and Application*, volume 4911 of *Lecture Notes in Artificial Intelligence*, pp. 222–243. Springer. DOI: 10.1007/978-3-540-78652-8_1. 33, 37, 42, 48, 61

Poole, D., Buchman, D., Natarajan, S., and Kersting, K. (2012). Aggregation and population growth: The relational logistic regression and Markov logic cases. In *Working Notes of the Workshop on Statistical Relational AI (StarAI@UAI-12)*. 55, 57

Poole, D. and Crowley, M. (2013). Cyclic causal models with discrete variables: Markov chain equilibrium semantics and sample ordering. In *Proc. of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*. 74

Poole, D., Smyth, C., and Sharma, R. (2009). Ontology design for scientific theories that make probabilistic predictions. *IEEE Intelligent Systems*, 24(1): 27–36. DOI: 10.1109/mis.2009.15. 58, 62

Poole, D.L. and Mackworth, A.K. (2010). *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, New York, NY. DOI: 10.1017/cbo9780511794797. 17, 33

Poon, H. and Domingos, P. (2008). Joint unsupervised coreference resolution with Markov logic. In *Proc. of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*. DOI: 10.3115/1613715.1613796. 11

Poon, H. and Domingos, P. (2009). Unsupervised semantic parsing. In *Proc. of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP-09)*. DOI: 10.3115/1699510.1699512. 11

Poon, H., Domingos, P., and Sumner, M. (2008). A general method for reducing the complexity of relational inference and its application to MCMC. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*. 87

Poon, H. and Domingos, P.M. (2007). Joint inference in information extraction. In *Proc. of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*. 11

Powell, W.B. (2010). Merging AI and OR to solve high-dimensional stochastic optimization problems using approximate dynamic programming. *INFORMS Journal on Computing*, 22(1): 2–17. DOI: 10.1287/ijoc.1090.0349. 127

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons. 124

Quinlan, J.R. (1990). Learning logical definitions from relations. *Machine Learning*, 5: 239–266. DOI: 10.1007/bf00117105. 101

Ramon, J., Driessens, K., and Croonenborghs, T. (2007). Transfer learning in reinforcement learning problems through partial policy recycling. In *Proc. of the 18th European Conference on Machine Learning (ECML-07)*, pp. volume 4701 of *Lecture Notes in Computer Science*, pp. 699–707. Springer. DOI: 10.1007/978-3-540-74958-5_70. 128

Ravkic, I., Ramon, J., and Davis, J. (2015). Learning relational dependency networks in hybrid domains. *Machine Learning*, 100(2–3): 217–254. DOI: 10.1007/s10994-015-5483-2. 119, 136

Reichenbach, H. (1949). *The Theory of Probability*. University of California Press. 105

Richards, B. and Mooney, R. (1992). Learning relations by pathfinding. *Proc. of the 10th National Conference on Artificial Intelligence (AAAI-92)*. 116

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62: 107–136. DOI: 10.1007/s10994-006-5833-1. 13, 34, 35, 36, 37, 48, 49, 52

Riedel, S. and Clarke, J. (2006). Incremental integer linear programming for non-projective dependency parsing. In *Proc. of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP-06)*. DOI: 10.3115/1610075.1610095. 130

Riedel, S., Smith, D., and McCallum, A. (2012). Parse, Price and Cut–Delayed Column and Row Generation for Graph Based Parsers. In *Proc. of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-12)*. 130

Riedel, S. and McCallum, A. (2011). Fast and robust joint models for biomedical event extraction. In *Proc. of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-11)*. 11

Rush, A. and Collins, M. (2012). A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45: 305–362. 129

Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, 3rd ed., Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ. DOI: 10.5860/choice.33-1577. 17

Russell, S. (2015). Unifying logic and probability. *Communications of ACM*, 58(7): 88–97. 58, 61

Sang, T., Beame, P., and Kautz, H. (2005). Solving Bayesian networks by weighted model counting. In *Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05)*. 51

Sanner, S. and Boutilier, C. (2009). Practical solution techniques for first order MDPs. *Artificial Intelligence*, 173: 748–788. DOI: 10.1016/j.artint.2008.11.003. 125, 127, 128, 129

Sanner, S. and Kersting, K. (2010a). Symbolic dynamic programming. In C. Sammut and G. Webb (Eds.), *Encyclopedia of Machine Learning*, pp. 946–954. Springer. DOI: 10.1007/978-0-387-30164-8. 125

Sanner, S. and Boutilier, C. (2007). Approximate solution techniques for factored first-order MDPs. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*. 127

Sanner, S. and Kersting, K. (2010b). Symbolic dynamic programming for first-order POMDPs. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. 125

Sarjant, S., Pfahringer, B., Driessens, K., and Smith, T. (2011). Using the online cross-entropy method to learn relational policies for playing different games. In *Proc. of the 2011 IEEE Conference on Computational Intelligence and Games (CIG-11)*. DOI: 10.1109/cig.2011.6032005. 128

Sarkhel, S., Venugopal, D., Singla, P., and Gogate, V. (2014). Lifted MAP inference for Markov logic networks. In *Proc. of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS-14)*, pp. 859–867. Volume 33 of the JMLR Proceedings. 90

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In L. Sterling (Ed.), *Proc. of the 12th International Conference on Logic Programming (ICLP-95)*, pp. 715–729. MIT Press. 13, 34, 107, 111

Sato, T. and Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*. 34, 37, 42, 52, 108, 119

Sato, T. and Kameya, Y. (2000). A Viterbi-like algorithm and EM learning for statistical abduction. In R. Dybowski, J. Myers, and S. Parsons (Eds.), *Workshop Notes of the UAI-00 Workshop on Fusion of Domain Knowledge with Data for Decision Support*. 111

Sato, T. and Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15: 391–454. 111

Sato, T. and Kameya, Y. (2008). New advances in logic-based probabilistic modeling by PRISM. In L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton (Eds.), *Probabilistic Inductive Logic Programming: Theory and Application*, volume 4911 of *Lecture Notes in Artificial Intelligence*, pp. 118–155. Springer. DOI: 10.1007/978-3-540-78652-8_1. 34, 38, 42

Schiegg, M., Neumann, M., and Kersting, K. (2012). Markov logic mixtures of gaussian processes: Towards machines reading regression data. In *Proc. of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, pp. 1002–1011. Volume 22 of the JMLR Proceedings. 11, 136

Schulte, O. and Khosravi, H. (2012). Learning graphical models for relational data via lattice search. *Machine Learning*, 88(3): 331–368. DOI: 10.1007/s10994-012-5289-4. 119

Schulte, O., Khosravi, H., and Man, T. (2012). Learning directed relational models with recursive dependencies. *Machine Learning*, 89(3): 299–316. DOI: 10.1007/s10994-012-5308-5. 115

Schulte, O., Qian, Z., Kirkpatrick, A.E., Yin, X., and Sun, Y.L. (2014). Fast learning of relational dependency networks. *CoRR*, abs/1410.7835. Extended version of the version presented at the International Conference on Inductive Logic Programming (ILP-2014). 119

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6(2): 461–464. DOI: 10.1214/aos/1176344136. 96

Segal, E., Battle, A., and Koller, D. (2003). Decomposing gene expression into cellular processes. In *Proc. Pacific Symposium on Biocomputing (PSB-03)*, pp. 89–100. World Scientific. DOI: 10.1142/9789812776303_0009. 10

Segal, E., Taskar, B., Gasch, A., Friedman, N., and Koller, D. (2001). Rich probabilistic models for gene expression. *Proc. of the 9th International Conference on Intelligent Systems for Molecular Biology (ISMB-01) Bioinformatics*, 17 (Suppl 1): S243–52. DOI: 10.1093/bioinformatics/17.suppl_1.s243. 10

Sen, P., Deshpande, A., and Getoor, L. (2008). Exploiting shared correlations in probabilistic databases. *Proc. of the VLDB Endowment*, 1: 809–820. DOI: 10.14778/1453856.1453944. 87, 88

Sen, P., Deshpande, A., and Getoor, L. (2009). Bisimulation-based approximate lifted inference. In *Proc. of the 25th Conference on Uncertainty in Artificial Intelligence (UAI-09)*. 88

Shafer, G. and Shenoy, P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2: 327–352. DOI: 10.1007/bf01531015. 51, 66

Shapiro, E. (1983). *Algorithmic Program Debugging*. MIT Press. 117

Sharma, R. and Poole, D. (2005). Probability and equality: A probabilistic model of identity uncertainty. In *Proc. of the 18th Canadian Conference on Artificial Intelligence*. DOI: 10.1007/11424918_24. 59

Shavlik, J. and Natarajan, S. (2009). Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *Proc. of the 21st International Joint Conference on Artifical Intelligence (IJCAI-09)*. 88, 137

Silva, R., Chu, W., and Ghahramani, Z. (2007). Hidden common cause relations in relational learning. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*. 34

Simon, H. (1996). *The Sciences of the Artificial*, 3rd ed., MIT Press, Cambridge, MA. 47

Singla, P. and Domingos, P. (2008). Lifted first-order belief propagation. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*. 81, 88

Singla, P., Nath, A., and Domingos, P. (2010). Approximate lifted belief propagation. In *Working Notes of the Workshop on Statistical Relational Artificial Intelligence (StarAI@AAAI-10)*. 88

Singla, P. and Domingos, P.M. (2006). Entity resolution with Markov logic. In *Proc. of the 6th IEEE International Conference on Data Mining (ICDM-06)*. DOI: 10.1109/icdm.2006.65. 11

Singla, P. and Domingos, P.M. (2007). Markov logic in infinite domains. In *Proc. of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*. 34, 37

Smith, B. (2003). Ontology. In L. Floridi (Ed.), *Blackwell Guide to the Philosophy of Computing and Information*, pp. 155—166. Oxford: Blackwell. DOI: 10.1002/9780470757017. 62

Song, Y.C., Kautz, H.A., Allen, J.F., Swift, M.D., Li, Y., Luo, J., and Zhang, C. (2013). A Markov logic framework for recognizing complex events from multimodal data. In *Proc. of the International Conference on Multimodal Interaction (ICMI-13)*, pp. 141–148. DOI: 10.1145/2522848.2522883. 12

Sowa, J.F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA. 45

Sowa, J.F. (2011). Future directions for semantic systems. In A. Tolk and L.C. Jain (Eds.), *Intelligence-based Software Engineering*, pp. 23–47. Springer Verlag. DOI: 10.1007/978-3-642-17931-0. 62

Sra, S., Nowozin, S., and Wright, S. (Eds.) (2011). *Optimization for Machine Learning*. MIT Press. 129

Srinivasan, A., Muggleton, S., King, R., and Sternberg, M. (1996). Theories for mutagenicity: a study of first-order and feature-based induction. *Artificial Intelligence*, 85: 277–299. DOI: 10.1016/0004-3702(95)00122-0. 102

Sterling, L. and Shapiro, E. (1986). *The Art of Prolog: Advanced Programming Techniques*. The MIT Press. 25

Tadepalli, P., Givan, R., and Driessens, K. (2004). Relational reinforcement learning: An overview. In *Proc. of the ICML Workshop on Relational Reinforcement Learning*. 128

Taghipour, N. and Davis, J. (2012). Generalized counting for lifted variable elimination. In *Working Notes of the Workshop on Statistical Relational AI (StarAI@UAI-12)*. DOI: 10.1007/978-3-662-44923-3_8. 82

Taghipour, N., Fierens, D., Van den Broeck, G., Davis, J., and Blockeel, H. (2013). Completeness results for lifted variable elimination. In *Proc. of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS-13)*, pp. 572–580. Volume 31 of the JMLR Proceedings. 82

Talbott, W. (2008). Bayesian epistemology. In E.N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy*. CSLI, Stanford University. DOI: 10.1145/379437.379789. 17

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-Margin Markov Networks. In S. Thrun, L. Saul, and B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems 16 (NIPS-04)*. 13

Tenenbaum, J., Kemp, C., Griffiths, T., and Goodman, N. (2011). How to grow a mind: Statistics, structure, and abstraction. *Science*, 331: 1279–1285. DOI: 10.1126/science.1192788. 11

Tenorth, M., Klank, U., Pangercic, D., and Beetz, M. (2011). Web-enabled robots—robots that use the web as an information resource. *IEEE Robotics and Automation Magazine*, 18. 10

Tenorth, M. and Beetz, M. (2013). Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *International Journal of Robotics Research*, 32(5): 566–590. DOI: 10.1177/0278364913481635. 12

Thon, I., Landwehr, N., and De Raedt, L. (2011). Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82(2): 239–272. DOI: 10.1007/s10994-010-5213-8. 135

Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*. 12

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27: 1134–1142. DOI: 10.1145/1968.1972. 100

Van den Broeck, G. (2011). On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems 24 (NIPS-11)*. 77, 83

Van den Broeck, G., Choi, A., and Darwiche, A. (2012). Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proc. of the 28th Conference on Uncertainty in Artificial Intelligence (UAI-12)*. 90

Van den Broeck, G. and Davis, J. (2012). Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proc. of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*. 87

Van den Broeck, G., Thon, I., van Otterlo, M., and De Raedt, L. (2010). DTProbLog: A decision-theoretic probabilistic prolog. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. 125, 128

Van den Broeck, G., Meert, W., and Darwiche, A. (2014). Skolemization for weighted first-order model counting. In *Proc. of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR-14)*. 82

Van den Broeck, G., Taghipour, N., Meert, W., Davis, J., and De Raedt, L. (2011). Lifted probabilistic inference by first-order knowledge compilation. In *Proc. of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, pp. 2178–2185. 53, 80, 82, 87

Van Haaren, J., Van den Broeck, G., Meert, W., and Davis, J. (2015). Lifted generative learning of Markov logic networks. *Machine Learning*. In press. DOI: 10.1007/s10994-015-5532-x. 110, 116

van Otterlo, M. (2009). *The Logic of Adaptive Behavior - Knowledge Representation and Algorithms for Adaptive Sequential Decision Making under Uncertainty in First-Order and Relational Domains*. IOS Press. 128

Vennekens, J., Denecker, M., and Bruynooghe, M. (2009). CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(03): 245-308. DOI: 10.1017/s1471068409003767. 38, 42

Vennekens, J., Verbaeten, S., and Bruynooghe, M. (2004). Logic programs with annotated disjunctions. In *Proc. of the 20th International Conference on Logic Programming (ICLP-04)*. volume 3132 of *Lecture Notes in Computer Science*, pp. 431-445. Springer. DOI: 10.1007/978-3-540-27775-0_30. 42

Venugopal, D., Chen, C., Gogate, V., and Ng, V. (2014). Relieving the computational bottleneck: Joint inference for event extraction with high-dimensional features. In *Proc. of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP-14)*. DOI: 10.3115/v1/d14-1090. 7

Venugopal, D. and Gogate, V. (2014). Evidence-based clustering for scalable inference in Markov logic. In *Proc. of the European Conference in Machine Learning and Knowledge Discovery in Databases (ECML PKDD-14)*, volume 8726 of *Lecture Notes in Computer Science*, pp. 258–273. DOI: 10.1007/978-3-662-44845-8_17. 87, 88

Venugopal, D., Sarkhel, S., and Gogate, V. (2015). Just count the satisfied groundings: Scalable local-search and sampling based inference in MLNs. In *Proc. of the 29th Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*. 137

Verbeke, M., Asch, V.V., Morante, R., Frasconi, P., Daelemans, W., and De Raedt, L. (2012). A statistical relational learning approach to identifying evidence based medicine categories. In *Proc. of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL-12)*, pp. 579–589. 11

Wang, C., Joshi, S., and Khardon, R. (2008). First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research*, 31: 431–472. 125, 127

Wang, C. and Khardon, R. (2010). Relational partially observable MDPs. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*. 125

Wang, W.Y., Mazaitis, K., Lao, N., and Cohen, W.W. (2015). Efficient inference and learning in a large knowledge base - reasoning with extracted information using a locally groundable first-order probabilistic logic. *Machine Learning*, 100(1): 101–126. DOI: 10.1007/s10994-015-5488-x. 88, 110

Weiss, J., Natarajan, S., Peissig, P., McCarty, C., and Page, D. (2012). Statistical relational learning to predict primary myocardial infarction from electronic health records. In *Proc. of the 24th Innovative Applications of Artificial Intelligence Conference (IAAI-12)*. 12

Wellner, B., McCallum, A., Peng, F., and Hay, M. (2004). An integrated, conditional model of information extraction and coreference with application to citation matching. In *Proc. Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pp. 593–601. 60

Williams, R. and Zipser, D. (1995). Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In *Back-propagation: Theory, Architectures and Applications*, pp. 433–486. Lawrence Erlbaum Associates, Hillsdale, NJ. 110

Wrobel, S. (1996). First Order Theory Refinement. In L. De Raedt (Ed.), *Advances in Inductive Logic Programming*. IOS Press. 117

Wu, J., Williams, K., Chen, H.H., Khabsa, M., Caragea, C., Ororbia, A., Jordan, D., and Giles, C.L. (2014). CiteseerX: AI in a digital library search engine. In *Proc. of the 26th Conference on Innovative Applications of Artificial Intelligence (IAAI-14)*. 60

Xiang, R. and Neville, J. (2011). Relational learning with one network: An asymptotic analysis. In *Proc. of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, pp. 779–788. Volume 15 of the JMLR Proceedings. 112

Xu, Z., Tresp, V., Rettinger, A., and Kersting, K. (2009a). Social network mining with nonparametric relational models. In *Advances in Social Network Mining and Analysis*, . volume 5498 of *Lecture Notes in Computer Science*, pp. 77–96. DOI: 10.1007/978-3-642-14929-0_5. 11

Xu, Z., Tresp, V., Yu, K., and Kriegel, H.P. (2006). Infinite hidden relational models. In *Proc. of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI-06)*. 34

Xu, Z., Kersting, K., and Joachims, T. (2010). Fast active exploration for link-based preference learning using Gaussian processes. In *Proc. of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD-10)*. volume 6323 of *Lecture Notes in Computer Science*, pp. 499–514. DOI: 10.1007/978-3-642-15939-8_32. 11

Xu, Z., Kersting, K., and Tresp, V. (2009b). Multi-relational learning with Gaussian processes. In *Proc. of the 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pp. 1309–1314. 34

Yih, W. and Roth, D. (2007). Global inference for entity and relation identification via a linear programming formulation. In L. Getoor and B. Taskar (Eds.), *An Introduction to Statistical Relational Learning*. MIT Press. 130

Yoshikawa, K., Riedel, S., Asahara, M., and Matsumoto, Y. (2009). Jointly identifying temporal relations with Markov logic. In *Proc. of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pp. 405–413. DOI: 10.3115/1687878.1687936. 11

Yu, K. and Chu, W. (2007). Gaussian process models for link analysis and transfer learning. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*. 34

Yu, K., Chu, W., Yu, S., Tresp, V., and Xu, Z. (2006). Stochastic relational models for discriminative link prediction. In *Advances in Neural Information Processing Systems 19 (NIPS-06)*. 34

Zettlemoyer, L.S., Pasula, H.M., and Kaelbling, L.P. (2007). Logical particle filtering. In *Proc. of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*. 87

Zhang, N.L. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Proc. 10th Canadian Conference on Artificial Intelligence*. 51, 66, 82

Authors' Biographies

LUC DE RAEDT

Luc De Raedt is a full professor of computer science at the KU Leuven (Belgium), where he is director of the Lab for Declarative Languages and Artificial Intelligence, and where he also obtained his Ph.D. He is also a former professor of computer science of the Albert-Ludwigs-University Freiburg (Germany) and chair of its lab for Natural Language Processsing and Machine Learning.

Luc De Raedt's research interests are in artificial intelligence, machine learning, and data mining, as well as their applications. He is currently working on probabilistic logic learning (sometimes called statistical relational learning), which combines probabilistic reasoning methods with logical representations and machine learning, the integration of constraint programming with data mining and machine learning principles, the development of programming languages for machine learning, and analyzing graph and network data. He is also interested in applications of these methods to chemo- and bio-informatics, to natural language processing, vision, robotics, and action and activity learning. He was program (co)-chair of the 7th ECML Machine Learning (1994, Catania, Sicily), the 5th ILP (1995, Leuven, Belgium), the first ECMLPKDD (2001, Freiburg, Germany), the 22nd ICML Learning (2005, Bonn, Germany) and the 20th ECAI (2012, Montpellier, France). He is an area/action editor of TPLP, JMLR, MLJ, AIJ, and formerly of JAIR. He is also a member of the editorial boards of NGC, AI Communications, Informatica, DMKD, and the *Journal of Applied Logic*. He was an elected and founding member of the board of the International Machine Learning Society from 2004–2011. In 2005, he was elected as an ECCAI fellow and four of his students have won the ECCAI dissertation award for the best European dissertation in AI.

KRISTIAN KERSTING

Kristian Kersting is an associate professor in the Computer Science Department at the Technical University of Dortmund, Germany. He received his Ph.D. from the University of Freiburg, Germany in 2006 and moved to the Fraunhofer IAIS and the University of Bonn using a Fraunhofer ATTRACT Fellowship in 2008 after a PostDoc at MIT, Cambridge, MA, U.S. Before moving to the TU Dortmund University in 2013, he was appointed Assistant Professor for Spatio-Temporal Patterns in Agriculture at the University of Bonn in 2012. Additionally, he was adjunct assistant professor at the Medical School of the Wake Forest University, Winston-Salem, NC, U.S., in 2012. His main research interests are data mining, machine learning, and statistical relational artificial intelligence. He has published over 120 peer-reviewed papers and received the

ECCAI Dissertation Award 2006, the ECML Best Student Paper Award in 2006, the ACM SIGSPATIAL GIS Best Poster Award in 2011, and the AAAI-2013 Outstanding PC Member Award. He has given several tutorials at top conferences and co-chaired BUDA, CMPL, CoLISD, MLG, SRL, and StarAI, as well as the AAAI Student Abstract track and the Starting AI Research Symposium (STAIRS). Together with Stuart Russell, Leslie Kaelbling, Alon Halevy, Sriraam Natarajan, and Lilyana Mihalkova he cofounded the international workshop series on Statistical Relational AI (StarAI). He served as area chair/senior PC for several top conference and co-chaired ECML PKDD 2013, the premier European venue for Machine Learning and Data Mining. Currently, he is an action editor of *AIJ, DAMI, JAIR*, and *MLJ* as well as on the editorial board of NGC.

SRIRAAM NATARAJAN

Sriraam Natarajan is an assistant professor at Indiana University. He was previously an assistant professor at Wake Forest School of Medicine, a post-doctoral research associate at University of Wisconsin-Madison, and graduated with his Ph.D. from Oregon State University. His research interests lie in the field of artificial intelligence, with emphasis on machine learning, statistical relational learning and AI, reinforcement learning, graphical models, and biomedical applications. He has received the Young Investigator award from U.S. Army Research Office. He is the organizer of the key workshops in the field of Statistical Relational Learning and has co-organized the AAAI 2010, the UAI 2012, AAAI 2013, and AAAI 2014 workshops on Statistical Relational AI (StarAI), ICML 2012 Workshop on Statistical Relational Learning, and the ECML PKDD 2011 and 2012 workshops on Collective Learning and Inference on Structured Data (Co-LISD). He is also the co-chair of the AAAI student abstract and posters at AAAI 2014 and AAAI 2015.

DAVID POOLE

David Poole is a professor of computer science at the University of British Columbia. He has a Ph.D. from the Australian National University. He is known for his work on assumption-based reasoning, diagnosis, relational probabilistic models, combining logic and probability, algorithms for probabilistic inference, representations for automated decision making, probabilistic reasoning with ontologies, and semantic science. He is a co-author of a new AI textbook, *Artificial Intelligence: Foundations of Computational Agents* (Cambridge University Press, 2010), co-author of an older AI textbook, *Computational Intelligence: A Logical Approach* (Oxford University Press, 1998), co-chair of AAAI-10 (twenty-Fourth AAAI Conference on Artificial Intelligence), and co-editor of the *Proceedings of the Tenth Conference in Uncertainty in Artificial Intelligence* (Morgan Kaufmann, 1994). He is a former associate editor of the *Journal of AI Research*, an the *AI Journal*, and the editorial board of *AI Magazine*. He is the chair of the Association for Uncertainty in Artificial Intelligence and is a Fellow of the Association for the Advancement Artificial Intelligence (AAAI). He is the winner of the Canadian AI Association (CAIAC) 2013 Lifetime Achievement Award. In the 2014–15 academic year he was a Leverhulme Trust visiting professor at the University of Oxford.

Index