12

A Survey of First-Order Probabilistic Models

Rodrigo de Salvo Braz*, Eyal Amir, and Dan Roth

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Summary. There has been a long standing division in Artificial Intelligence between logical and probabilistic reasoning approaches. While probabilistic models can deal well with inherent uncertainty in many real-world domains, they operate on a mostly propositional level. Logic systems, on the other hand, can deal with much richer representations, especially first-order ones, but treat uncertainty only in limited ways. Therefore, an integration of these types of inference is highly desirable, and many approaches have been proposed, especially from the 1990s on. These solutions come from many different subfields and vary greatly in language, features and (when available at all) inference algorithms. Therefore their relation to each other is not always clear, as well as their semantics. In this survey, we present the main aspects of the solutions proposed and group them according to language, semantics and inference algorithm. In doing so, we draw relations between them and discuss particularly important choices and tradeoffs.

For decades after the field of Artificial Intelligence (AI) was established, its most prevalent form of representation and inference was logical, or at least symbolic representations that were in a deeper sense equivalent to a fragment of logic. While highly expressive, this type of model lacked a sophisticated treatment of *degrees* of uncertainty, which permeates real-world domains, especially the ones usually associated with intelligence, such as language, perception and common sense reasoning.

In time, probabilistic models became an important part of the field, incorporating probability theory into reasoning and learning AI models. Since the 1980s the field has seen a surge of successful solutions involving large amounts of data processed from a probabilistic point of view, applied especially to Natural Language Processing and Pattern Recognition.[1]

* Currently at the Computer Science Division of University of California, Berkeley.

[1] Strictly speaking, this tendency has not been only probabilistic, including machine learning methods such as neural networks that did not claim to be modeling probabilities. However, a link to probabilities can usually be found and the methods are used in similar ways.

This success, however, came with a price. Typically, probabilistic models are less expressive and flexible than logical or symbolic systems. Usually, they involve propositional, rather than first-order representations. When required, more expressive, higher level representations are obtained by ad hoc manipulations of lower level, propositional systems.

Starting in the 1970s but having greatly increased from the 1990s on, a line of research sought to integrate those two important modes of reasoning. In this chapter we give a survey of this research, and try to show some general lines separating different approaches.

We have roughly divided this research in different stages. The 1970s and 1980s saw great interest in expert systems [1, 2]. As these systems were applied to real-world domains, coping with uncertainty became more desirable, giving rise to the certainty factors approach, which uses rules with attached numbers (representing degrees of certainty) that get propagated to conclusions during inference.

Certainty factors systems did not have clear semantics, and often produced surprising and nonintuitive results [3]. The search for clearer semantics for rules with varying certainty gave rise, among other things, to approaches such as Bayesian Networks. These however were essentially propositional, and thus had much less expressivity than logic systems.

The search for clear semantics of probabilities in logic systems resulted in works such as Nilsson [4], Bacchus [5] and Halpern [6], which laid out the basic theoretic principles supporting probabilistic logic. These works, however, did not include efficient inference algorithms.

Works aiming at efficient inference algorithms for first-order probabilistic inference (FOPI) can be divided in two groups, which Pearl [3] calls *extensional* and *intensional* systems. In the first one, statements in the language are more procedural in nature, standing for licenses for propagating truth values that have been generalized from true or false to a gray scale of varying degrees of certainty. In the second group, statements place restrictions on a probability distribution on possible worlds. They do not directly correspond to computing operations, nor can they typically be taken into account without regard to other rules (that is, inference is not completely modular). Efficient algorithms have to be devised for these languages that preserve their semantics while doing better than considering the entire model at every step.

Among intensional models, there are further divisions regarding the type of algorithm proposed. One group proposes inference rules similar to the ones used in first-order logic inference (for example, modus ponens). A second one computes, in more or less efficient manners, the possible derivations of a query given a model. A third one uses sampling to answer queries about a model. A fourth and more prevalent group constructs a (propositional) graphical model (Bayesian or Markov networks, for example) that answers queries, and uses general graphical model inference algorithms for solving them. Finally, a fifth one proposes *lifted* algorithms that directly operate on first-order representations in order to derive answers to queries.

We now present these stages in more detail.

12.1 Expert Systems and Certainty Factors

Expert systems are based on rules meant to be applied to existing facts, producing new facts as conclusions [1]. Typically, the context is a deterministic one in which facts and rules are assumed to be certain. Uncertainties from real-world applications are dealt with during the modeling stage where necessary (and often heavy-handed) simplifications are performed.

Certainty factors were introduced for the purpose of allowing uncertain rules and facts, making for more direct and accurate modeling. A rule $(A \leftarrow B) : c_1$, with $c_1 \in [0, 1]$, indicates that we can conclude A with a degree of certainty of $c_1 \times c_2$, if B is known to be true with a degree of certainty $c_2 \in [0, 1]$. Given a collection of rules and facts, inference is performed by propagating certainties in this fashion. There are also combination rules for the cases when more than one rule provide certainty factors for the same literal.

A paradigmatic application of certainty factors is the system MYCIN [7], an expert system dedicated to diagnosing diseases based on observed symptoms. Clark & McCabe [8] describe how to use Prolog with predicates containing an extra argument representing its certainty and being propagated accordingly. Shapiro [9] describes a Prolog interpreter that does the same but in a way implicit in the interpreter and language, rather than as an extra argument.

One can see that certainty factors have a probabilistic flavor to them, but formally they are not taken to be probabilistic. This is for good reason: should we interpret them as probabilities, results would be inconsistent with probability theory. Heckerman [10] and Lucas [11] discuss situations in which certainty factor computations can and cannot be correctly interpreted probabilistically. One reason they cannot is the incorrect treatment of bidirectional inference: two certainty factor rules $(A \leftarrow B) : c_1$ and $B : c_2$ imply nothing about inference from A to B, while $P(A|B)$ and $P(B)$ do place constraints on $P(B|A)$. These problems are further discussed in Pearl [3].

12.2 Probabilistic Logic Semantics

The semantic limitations of certainty factors is one of the motivations for defining precise semantics for probabilistic logics, but such investigations date from at least as far back as Carnap [12].

One of the most influential AI works in this regard is Nilsson [4] (a similar approach is given by Hailperin [13]). Nilsson establishes a systematic way of determining the probabilities of logic sentences in a query set, given the probabilities of logical sentences in an evidence set. To be more precise, the method determines *intervals* of probabilities to the query sentences, since in principle the evidence set may be consistent with an entire range of point probabilities for them. For example, knowing that A is true with probability 0.2 and B with probability 0.6 means that $A \wedge B$ is true with probability in $[0, 0.2]$, depending on whether A and B are mutually exclusive, or $A \rightarrow B$, or anything in between.

Given a set of sentences L, Nilsson considers the equivalence classes of possible worlds that assign the same truth values to the sentences in L (that is, as far as L is concerned, all possible worlds in the same class are the same). Formally, Nilsson's system is based on the following linear problem:

$$\begin{aligned} \Pi &= VP \\ 0 \leq \Pi_j &\leq 1 \\ 0 \leq P_i &\leq 1 \\ \sum_i P_i &= 1 \end{aligned}$$

where Π is the vector of probabilities of sentences in both query and evidence sets, P the vector of probabilities of possible worlds equivalence classes, and V is a matrix with $V_{ij} = 1$ if sentence j is true in possible world set i, and 0 otherwise. The probabilities of sentences in the knowledge base are incorporated as constraints in this system as well, and linear programming techniques can be used to determine the probability of novel sentences. However, as Nilsson points out, the problem becomes intractable even with a modest number of sentences, since all possible worlds equivalence classes need to be enumerated and this is an intractable problem. Therefore this framework cannot be directly used in practice.

Placing the probabilities on the possible worlds, as does Nilsson, makes it easy to express subjective probabilities such as "Tweety flies with probability 0.9" (that is, the sum of probabilities of all possible worlds in which Tweety flies is 0.9). However, probabilistic knowledge can also express statistical facts about the domain such as "90% of birds fly" (which says that, in each possible world, 90% of birds fly). Bacchus [5] provides an elaborate probabilistic logic semantics that includes both types of probabilistic knowledge, making it possible to use both statements above, as well as statements mixing them, such as "There is a probability of 0.8 that 90% of birds fly." He also discusses the interplay between the two types, namely the question of when it is correct to use the fact that "90% of birds fly" in order to assume that "a randomly chosen bird flies with probability 0.9," a topic that has both formal and philosophical aspects. Halpern [6] elaborates on the axiomatization of Bacchus, taking probabilities to be real numbers (Bacchus did not), and is often cited as a reference for this semantics with two types of probabilities. In subsequent work, the subjective type probability has been much more developed and used, and is also the type involved in propositional graphical models.

Fagin, Halpern, Meggido [14] present a logic to reason *about* probabilities, including their addition and multiplication by scalars. Other works discussing the semantics of probabilities on first-order structures are [15, 16, 17].

12.3 Extensional Approaches

Somewhat parallel to the works on the semantics of probabilistic logic, a different line of research proposed logic reasoning systems incorporating uncertainty in the explicit form of probabilities (as opposed to certainty factors). These systems often stem from the fields of logic programming and deductive databases, and fit into the category described by [3] as *extensional* systems, that is, systems in which rules work as "procedural licenses" for a computation step instead of a constraint on possible probability distributions. Most of these systems operate on a collection of rules or clauses that propagate generalized truth values (typically, a value or interval in $[0, 1]$).

Kiefer and Li [18] provide a probabilistic interpretation and a fixpoint semantics to Shapiro [9]. Wüthrich [19] elaborates on their work, taking into account partial dependencies between clauses. For example, if each of them atoms a, b and c has a prior probability of 0.5 and we have two rules $p \leftarrow a \wedge b$ and $p \leftarrow b \wedge c$, Kiefer and Li will assume the rules independent and assign a probability $0.25 + 0.25 - 0.25 * 0.25 = 0.4375$ to p. Wüthrich's system, however, takes into account the fact that b is shared by the clauses and computes instead $0.25 + 0.25 - 0.5^3 = 0.375$ (that is, it avoids double counting of the case where the two rules fire at the same time, which occurs only when the three atoms are true at once).

One of the most influential works within the extensional approach is Ng and Subrahmanian [20]. Here, a logic programming system uses generalized truth values in the form of intervals of probabilities. They define probabilistic logic program as sets of *p-clauses* of the form

$$A : \mu \leftarrow F_1 : \mu_1 \wedge \ldots F_n : \mu_n,$$

where A is an atom, $F_1, \ldots, F_n$ are basic formulas (conjunctions or disjunctions) and $\mu, \mu_1, \ldots, \mu_n$ are probability intervals. A clause states that if the probability of each formula F_i is in μ_i, then the probability of A is in μ. For example, the clause

$$path(X, Y) : [0.8, 0.95] \leftarrow a(X, Z) : [1, 1] \wedge path(Z, Y) : [0.85, 1]$$

states that, if $a(X, Z)$ is certain (probability in interval $[1, 1]$ and therefore 1) and $path(Z, Y)$ has probability in $[0.85, 1]$, then $path(X, Y)$ has probability in $[0.8, 0.95]$. Probabilities of basic formulas F_i are determined from the probability intervals of their conjuncts (or disjuncts) by taking into account the possible correlations between them (similarly to what Nilsson does). The authors present a fixpoint semantics where clauses are repeatedly applied and probability intervals successively narrowed up to convergence. They also develop a model theory determining what models (sets of distributions on possible worlds) satisfy a probabilistic logic program, and a refutation procedure for querying a program.

Lakshmanan and Sadri [21] propose a system similar to Ngo and Subrahmanian, while keeping track of both the probability of each atom as well of its negation. Additionally, it uses configurable independence assumptions for different clauses, allowing the user to declare whether two atoms are independent, mutually exclusive, or even the lack of an assumption (as in Nilsson). Lakshmanan [22] separates the qualitative and quantitative aspects of probabilistic logic. Dependencies between atoms are declared in terms of the boolean truth values of a set of *support* atoms. Only later is a distribution assigned to the support atoms, consequently defining distributions on the remaining atoms as well. The main advantage of the approach is the possibility of investigating different total distributions, based on distributions on the support set, without having to recalculate the relationship between atoms and support set. The algorithms works in ways similar to Ngo and Haddawy [23] and Lakshmanan and Sadri [21], but propagates support set conditions rather than probabilities. Support sets are also a concept very similar to the *hypotheses* used in Probabilistic Abduction by Poole [24] (see next section).

12.4 Intensional Approaches

We now discuss intensional approaches to probabilistic logic languages, where statements (often in the form of rules) are interpreted as restrictions on a globally defined probability distribution. This probability distribution is over all possible worlds or, in other words, on assignments to the set of all possible random variables in the language. Statements typically pose constraints in the form of conditional probabilities, and often also as conditional independence relations. (As mentioned in Sect. 12.2, another possibility would be *statistical* constraints, but this has not been explored in any works to our knowledge.)

The algorithms in intensional approaches, when available, are arguably more complex than extensional approaches, since their steps do not directly correspond to the application of rules in the language and need to be consistent with the global distribution while being as local as possible (for efficiency reasons).

We cover five different types of intensional approaches: deduction rules, exhaustive computation of derivations, sampling, Knowledge Based Model Construction (KBMC) and Lifted inference.

12.4.1 Deduction Rules

Classical logic deduction systems often work by receiving a model specified in a particular language and using deduction rules to derive new statements (guaranteed to be true) from subsets of previous statements. Some work has been devoted to devising similar systems when the language is that of probabilistic logic.

This method is particularly challenging in probabilistic systems because probabilistic inference is not as modular as classical logical inference. For example, while the logical knowledge of $A \rightarrow B$ allows us to deduce B given that $A \wedge \varphi$ is true for any formula φ, knowing $P(B|A)$ in itself does not tell us *anything* about $P(B|A \wedge \varphi)$. In principle, one needs to consider *all* available knowledge when establishing the conditional probability of B. Classical logic reasoning shows a modularity that is harder to achieve in a probabilistic setting.

One way of making probabilistic inference more modular is to use knowledge about conditional independencies between random variables. If we know that B is independent of any other random variable given A, then we know that $P(B|A \wedge \varphi)$

is equal to $P(B|A)$ for any φ. This has been the approach of graphical models such as Bayesian and Markov networks [3], where independencies are represented by the structure of a graph over the set of random variables.

The computation steps of specific inference algorithms for graphical models (such as Variable Elimination [25]) could be cast as deduction rules, much like in classical logic. However this is not traditionally done, mostly because inference rules are typically described in a logic-like language and graphical models are not. When dealing with a *first-order* probabilistic logic language, however, this approach becomes more natural.

Luckasiewicz [26] uses inference rules for solving trees of probabilistic conditional constraints over basic events. These trees are similar to Bayesian networks, with each node being a random variable and each edge being labeled by a conditional probability table. However, these trees are not meant to encode independence assumptions. Besides, conditional probabilities can also be specified in intervals.

Frisch and Haddawy [27] present a set of inference rules for probabilistic propositional logic with interval probabilities. They characterize it as an anytime system since inference rules will increasingly narrow those intervals. They also provide more modular inference by allowing statements on conditional independencies of random variables, which are used by certain rules to derive statements based on local information.

Koller and Halpern [28] investigate the use of independence information for FOPI based on inference rules. They use this notion to discuss the issue of *substitution* in probabilistic inference. While substitution is fundamental to classical logic inference, it is not sound in general in a probabilistic context. For example, inferring $P(q(A)) = \frac{1}{3}$ given $\forall P(q(X)) = \frac{1}{3}$ is not sound. Consider three possible worlds w_1, w_2, w_3 containing the three objects o_1, o_2, o_3 each, where $q(o_i)$ is 1 in w_i and 0 otherwise. If each possible world has a probability $\frac{1}{3}$ of being the actual world, then $\forall P(q(X)) = \frac{1}{3}$ holds. However, if A refers to o_i in each w_i, then $P(q(A)) = 1$. While this problem can be solved by requiring constants to be rigid designators (that is, each of them refers to the same object in all worlds), the authors argue that this is too restrictive. Their solution is to use information on independence. They show that when the statements $\forall P(q(X)) = \frac{1}{3}$ and $x = A$ are independent, one can derive $P(q(A)) = \frac{1}{3}$. Finally, they discuss the topic of using statistical probabilities as a basis for subjective ones (the two types discussed by Bacchus [5] and Halpern [6]) based on independencies.

12.4.2 Exhaustive Computation of Derivations

Another type of intensional system is the one in which the available algorithms exhaustively compute the set of derivations or proofs for a query, in the same way proofs are found for queries in logic programming. However, while in logic programming it is often only necessary to find one proof for a certain query, in probabilistic models all proos will typically influence the query's result, and therefore need to be computed.

Riezler [29] presents a probabilistic account of Constraint Logic Programs (CLPs) [30]. In regular logic programming, the only constraints over logical variables are equational constraints coming from unification. CLPs generalize this by allowing other constraints to be stated over those variables. These constraints are managed by special-purpose constraint solvers as the derivation proceeds, and failure in satisfying a constraint determines failure of the derivation. Probabilistic Constraint Logic Programs (PCLPs) are a stochastic generalization of CLPs, where clauses are annotated with a probability and chosen for the expansion of a literal according to that probability, among the available clauses with matching heads. The probability of a derivation is determined by the product of probabilities associated to the stochastic choices. In fact, PCLPs are a generalization of Stochastic Context-Free Grammars (SCFGs) [31], the difference between them being that PCLP symbols have arguments in the form of logical variables with associated constraints while grammar symbols do not. For this reason, PCLP derivations can fail while SCFGs will always succeed. This presents a complication for PCLP algorithms because the probability has to be normalized with respect to the sum of *successful* derivations only. It also makes the use of efficient dynamic programming techniques such as the inside-outside algorithm [32] not adequate for PCLPs, forcing us to compute all possible derivations of a query. Riezler focuses on presenting an algorithm for learning the parameters of a PCLP from incomplete data, in what is a generalization of the Baum-Welch algorithm for HMMs [33].

Stochastic Logic Programs [34, 35] are very similar to PCLPs, restricting themselves to regular logic programming (e.g., Prolog). This line of work is more focused on the development of an actual system on top of a Prolog interpreter and to be used with Inductive Logic Programming techniques such as Progol [36]. Like Riezler, in [35] Cussens develops methods for learning parameters of SLPs using Improved Interative Scaling [37] and the EM algorithm [38].

Luckasiewicz [39] presents a form of Probabilistic Logic Programming that complements Nilsson's [4] approach. Nilsson considers all equivalence classes of possible worlds with respect to the given knowledge and builds a linear program in order to assign probabilities to sentences. Luckasiewicz essentially does the same by using logic programming for both determining the the equivalence classes and the linear program.

Baral et al. [40] use answer set logic programming to implement a powerful probabilistic logic language. Its distinguishing feature is the possibility of specifying observations and actions, with their corresponding implications with respect to causality, as studied by Pearl [41]. However, the implementation, using answer set Prolog, depends on determining all answer sets.

12.4.3 Sampling Approaches

Because building all drivations of a query given a program is very expensive, approximation solutions become an attractive alternative.

Sato [42] presents PRISM, a full-featured Prolog interpreter extended with probabilistic switches that can be used to encode probabilistic rules and facts.

These switches are special built-in predicates that randomly succeed or not, following a specific probability distribution. They can be placed in the body of a clause, which in consequence will succeed or not with the same distribution (when the rest of the body succeeds). Therefore, multiple executions of the program will yield different random results that can be used as samples. A query can then be answered by multiple executions which sample its possible outcomes. Sato also provides a way of learning the parameters of the switches by using a form of the EM algorithm [43].

BLOG [44] is a first-order language allowing the specification of a generative model on a first-order structure. It is similar in form to BUGS [45], a specification language for propositional generative models. The main distinction of BLOG is its open world assumption; it does not require that the number of objects in the world be set a priori, using instead a prior on this number and also keeping track of identities of objects with different names. BLOG computes queries by sampling over possible worlds.

12.4.4 Knowledge Based Model Construction

We now present the most prominent family of models in the field of FOPI models, Knowledge Based Model Construction (KBMC). These approaches work by generating a propositional graphical model from a first-order language specification that answers the query at hand. This construction is usually done in a way specific to the query, ruling out irrelevant portions of the graph so as to increase efficiency.

Many KBMC approaches use a first-order logic-like specification language, but some use different languages such as frame systems, parameterized fragments of Bayesian networks, and description logics. Some build Bayesian networks while others prefer Markov networks (and in one case, Dependency Networks [46]).

While KBMC approaches try to prune sections of underlying graphical models which are irrelevant to the current query, there is still potentially much wasted computation because they may replicate portions of the graph which require essentially identical computations. For example, a problem may involve many employees in a company, and the underlying graphical model will contain a distinct section with its own set of random variables for each of them (representing their properties), even though all these sections have essentially the same structure. Often the same computation will be repeated for each of those sections, while it is possible to perform it only once in a generalized form. Avoiding this waste is the object of Lifted First-Order Probabilistic Inference [47, 48], discussed in Sect. 12.4.5.

The most commonly referenced KBMC approach is that of Breese [49, 50], although Horsch and Poole [51] had presented a similar solution a year before. [49] defines a probabilistic logic programming language, with Horn clauses annotated by probabilistic dependencies between the clause's head and body. Once a query is presented, clauses are applied to it in order to determine the probabilistic dependencies relevant to it. These dependencies are then used to form a Bayesian network. Backward inference will generate the causal portion of the network relative to the query; forward inference creates the diagnostic part. The construction algorithm uses the evidence in order to decide when to stop expanding the network – there is no need to generate portions that are d-separated from the query by the evidence. In fact, this work covers not only Bayesian networks, but influence diagrams as well, including decision and utility value nodes.

There are many works similar in spirit to [49] and differing only in some details; for example, the already mentioned Horsch and Poole [51], which also uses mostly Horn clauses (it does allow for universal and existential quantifiers over the entire clause body though) as the first-order language. One distinction in this work, however, is the more explicit treatment of the issue of *combination functions*, used to combine distributions coming from distinct clauses with the same head. One example of a combination function is noisy-or [3], which assumes that the probability provided by a single clause is the probability of it making the consequent true regardless of the other clauses. Suppose we have clauses $A \leftarrow B$ and $A \leftarrow C$ in the knowledge base, the first one dictating a probability 0.8 for A when B is true and the second one dictating a probability 0.7 for A when C is true. Then the combination function noisy-or builds a Conditional Probability Table (CPT) with B and C as parents of A, with entries $P(A|B,C) = \{1-0.2\times 0.3, 1-0.8\times 0.3, 1-0.2\times 0.7, 1-0.8\times 0.7\} = \{0.94, 0.76, 0.86, 0.47\}$ for $\{(B=\top, C=\top), (B=\bot, C=\top), (B=\top, C=\bot), (B=\bot, C=\bot)\}$, respectively.

In first-order models, noisy-or and other combination functions are especially useful when a random variable has a varying number of parents, which makes its CPT impossible to represent by a fixed-dimensions table. A clause $p \leftarrow q(X)$, for example, determines that p depends on all instantiations of $q(X)$, that is, all instantiations of $q(X)$ are parents of p. However, the number of such instantiations depends on how many values X can take. Without knowing this number, the only way of having a general specification of p's CPT is to have a combination function on the instantiations of $q(X)$. In fact, even when this number is known it may be convenient to represent the CPT with a combination function for compactness sake.

Charniak and Goldman [52] expand a deductive database and truth maintenance system (TMS) in order to define a language for constructing Bayesian networks. The Bayesian networks come from the data-dependency network maintained by the TMS system, which is annotated with probabilities. There is also a notion of combination functions. The authors choose not to expand logical languages, justifying this choice by arguing that logic and probability do not correspond perfectly, the first being based on implication while the second on conditioning.

Poole [24] defines Probabilistic Abduction, a probabilistic logic language aimed at performing abduction reasoning. Probabilities are defined only for a set of predicates, called *hypotheses* (which is reminiscent of the support set in [22]), while the clauses themselves are deterministic. When a problem has naturally dependent hypotheses, one can redefine them as regular predicates and invent a new hypothesis to explain that dependence. While deterministic clauses can seem too restrictive, one can always get the effect of probabilistic rules by using hypotheses as a condition of the rule (like switches in Sato's PRISM [42]). The language also assumes that the bodies of clauses with the same head are mutually exclusive, and again this is not as restrictive as it might seem since clauses with non-mutually exclusive bodies can be rewritten as a different set of clauses satisfying this. As in other works in this section, the actual computation of probabilities is based on the construction of a Bayesian network. In [53], Poole extends Probabilistic Abduction for decision theory, including both utility and decision variables, as well as negation as failure.

Glesner and Koller [54] present a Prolog-like language that allows the declaration of facts about a Bayesian network to be constructed by the inference process. The computing mechanisms of Prolog are used to define the CPTs as well, so they are not restricted to tables, but can be computed on the fly. This allows CPTs to be defined as decision trees, for example, which provides a means of doing automatic pruning of the resulting Bayesian network – if the evidence provides information enough to make a CPT decision at a certain tree node, the descendants of that node, along with parts of the network relevant to those descendants only, do not need to be considered or built. The authors focus on flexible dynamic Bayesian networks that do not necessarily have the same structure at every time slice.

Haddawy [55] presents a language and construction method very similar to [51, 49]. However, he focuses on defining the semantics of the first-order probabilistic logic language directly, and independently of the Bayesian network construction, and proceeds to use it to prove the correctness of the construction method. Breese [49] had done something similar by defining the semantics of the knowledge base as an abstract Bayesian network which does not usually get built itself in the presence of evidence, and by showing that the Bayesian network actually built will give the same result as the abstract one.

Koller and Pfeffer [56] present an algorithm for learning the probabilities of noisy first-order rules used for KBMC. They use the EM algorithm applied to the Bayesian networks generated by the model, using incomplete data. This works in the same way as the regular Bayesian network parameter learning with EM, with the difference that many of the parameters in the generated networks are in fact *instances* of the same parameter in a first-order rule. Therefore, all updates on these parameters must be accumulated in the original parameter.

Jaeger [57] defines a language for specifying a Bayesian network whose nodes are the extensions of first-order predicates. In other words, each node is the assignment to the set *all* atoms of a certain predicate. Needless to say, inference in such a network would be extremely inefficient since each node would have an extremely large number of values. However, it offers the advantage of making the semantics of the language very clear (it is just the usual propositional Bayesian network semantics – the extension of a predicate is just a propositional variable with a very large number of values). The author proposes, like other approaches here, to build a regular Bayesian network (with a random variable per ground atom) for the purpose of answering specific queries. He also presents a sophisticated scheme for combination functions, including the possibility of their nesting.

Koller et al. [58, 59] define Probabilistic Relational Models (PRMs), a sharp depart from the logical-probabilistic models that had been proposed until then as solutions for FOPI models. Instead of adding probabilities to some logic-like language, the authors use the formalism of Frame Systems [60] as a starting point. The language of frames, similar also to relational databases, is less expressive than first-order logic, which is to the authors one of its main advantages since first-order logic inference is known to be intractable (which only gets worse when probabilities are added to the mix). By using a language that limits its expressivity to what is most needed in practical applications, one hopes to obtain more tractable inference, an argument commonly held in the Knowledge Representation community [61]. In fact, Pfeffer and Koller had already investigated adding probabilities to restricted languages in [62]. In that case, the language in question was that of description logics.

The language of Frame Systems consists of defining a set of objects described by *attributes* – binary predicates relating an object to a simple scalar value – or *relations* – binary predicates relating an object to another (or even self) object. PRMs add probabilities to frame systems by establishing distributions on attributes conditioned on other attributes (in the same object, or related object). In order to avoid declaring these dependencies for each object, this is done at a *scheme* level where classes, or template objects, stand for all instances of a class. This scheme describes the attributes of classes and the relations between them. Conditional probabilities are defined for attributes and can name the conditioning attributes via the relations needed to reach them.

As in the previous approaches, queries to PRMs are computed by generating an underlying Bayesian network. Given a collection of objects (a database *skeleton*) and the relationships between them, a Bayesian network is built with a random variable for each attribute in each object. The parents of these random variables in the network are the ones determined by the relations in the particular database, and the CPT filled with the values specified at the template level. An example of this process is shown in Fig. 12.1.

Note that the set of ancestors of attributes in the underlying network is determined by the relations from one object to another. One could imagine an attribute *rating* of an object representing a restaurant that depends on the attribute *training* of the object representing its chef (related to it by the relationship $chef$). In approaches following first-order representations, $chef$ would be a binary predicate, and each of its instances a random variable. As a result, the ancestors of *rating* would be the attributes *training* of all objects potentially linked to the restaurant by the relationship $chef$, plus the random variables standing for possible pairs in the relationship *cook* itself, resulting in a large (and thus expensive) CPT. PRMs avoid this when they take data with a defined structure where the assignment to relations such as *cook* is known; in this case, the random variables in the relationship $chef$ would not even be included in the network, and the attribute *rating* of each object would have a single

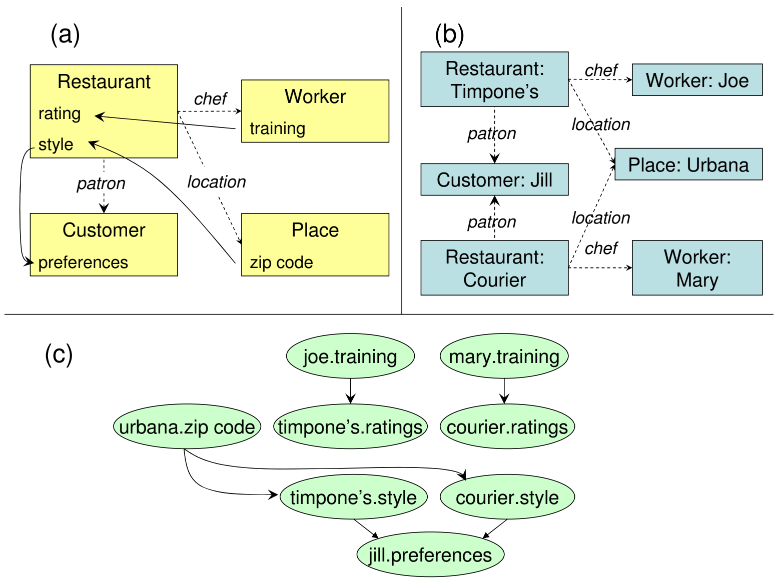


Fig. 12.1. (a) A PRM scheme showing classes of objects (rectangles), probabilistic dependencies between their attributes (full arrows) and relationships (dashed arrows). (b) A database skeleton showing a collection of objects, their classes and relationships. (c) The corresponding generated Bayesian network.

ancestor. When relationships are not fixed in advance, we have *structural uncertainty*, which was addressed by the authors in [63]. These papers have presented PRM learning of both parameters and structure (that is, the learning of the scheme level).

PRMs make use of Bayesian networks, a directed graphical model that brings a notion of causality. In relational domains it is often the case that random variables depend on each other without a clear notion of causality. Take for example a network of people linked by friendship relationships, with the attribute *smoker* for each person. We might want to state the first-order causal relationship $P(smoker(X)|friends(X,Y), smoker(Y))$ in such a model, but it would create cycles in the underlying Bayesian network (between each pair of *smoker* attributes such as $smoker(john)$ and $smoker(mary)$). For this reason, Relational Markov Networks (RMNs) [64] recast PRMs so they generate undirected graphical models (Markov networks) instead of Bayesian networks. In RMNs, dependencies are stated as first-order features that get instantiated into potential function on cliques of random variables, without a notion of causality or conditional probabilities. The disadvantage of it, however, is that learning in undirected graphical models is harder than in directed ones, involving a full inference step at each expectation step of the EM algorithm.

Relational Dependency Networks (RDNs) [65] provide yet another alternative to this problem. They are the first-order version of Dependency Networks

(DNs) (Heckerman, [46]), which use conditional probabilities but do not require acyclicity. Using directed conditional probabilities avoids the expensive learning of undirected models. However, DNs have the downside of conditional probabilities being no longer guaranteed consistent with the joint probability defined by their normalized product. Heckerman shows that, as the amount of training data increases, conditional probabilities in a DN will asymptotically converge to consistency. RDNs are sets of first-order conditional probabilities which are used to generate an underlying regular dependency network. These first-order conditional probabilities are typically learned from data by relational learners (Sect. 12.5). RDNs are implemented in Proximity, a well-developed, publicly available software package.

Kersting and DeRaedt [66] introduce Bayesian Logic Programs. This work's motivation is to provide a language which is as syntactically and conceptually simple as possible while preserving the expressive power of works such as Ngo and Haddawy [23], Jaeger [57] and PRMs [58]. According to the authors, this is necessary so one understands the relationship between all these approaches, and also the fundamental aspects of FOPI models.

Fierens at al [67] define Logical Bayesian Networks (LBNs). LBNs are very similar to Bayesian Logic Programs, with the difference of having both random variables and deterministic logical literals in their language. A logic programming inference process is run for the construction of the Bayesian network, during which logical literals are used, but since they are not random variables, they are not included in the Bayesian network. This addresses the same issue of fixed relationships discussed in the presentation of PRMs, that is, when a set of relationships is deterministically known, we can create random variable nodes in the Bayesian network with significantly fewer ancestors. In the BLPs and LBNs framework, this is exemplified by a rule such as:

$$rating(X) \leftarrow cook(X, Y), training(Y) \ .$$

which has an associated probability, declaring that a restaurant X's rating depends on their cook Y's training. In Bayesian Logic Programs, the instantiations of $cook(X, Y)$ are random variables (just like the instantiations of $rating(X)$ and $training(Y)$). Therefore, since we do not know a priori which Y makes $cook(timpone, Y)$ true, $rating(timpone)$ depends on all instantiations of $cook(timpone, Y)$ and $training(Y)$ and has all of them as parents in the underlying Bayesian network. If in the domain at hand the information of *cook* is deterministic, then this would be wasteful. We could instead determine Y such that $cook(timpone, Y)$, say $Y = joe$, and build the Bayesian network with only the relevant random variable $training(joe)$ as parent of $rating(timpone)$. This is precisely what LBNs do. In LBNs, one would define *cook* as a deterministic literal that would be reasoned about, but not included in the Bayesian network as a random variable. This in fact is even more powerful than the PRMs approach since it deals even with the situation where relationships are not directly given as data, but have to be reasoned about in a deterministic manner.

Santos Costa et al. [68] propose an elegant KBMC approach that smoothly leverages an already existing framework, Constraint Logic Programming (CLP). In regular logic programming, the only constraint over logical variables are equational constraints coming from unification. As explained in Sect. 12.4.2, CLP programs generalize this by allowing other constraints to be stated over those variables. These constraints are managed by special-purpose constraint solvers as the derivation proceeds, and failure in satisfying a constraint determines failure of the derivation. The authors leverage CLP by developing a constraint solver on probabilistic constraints expressed as CPTs, and simply plug it into an already existing CLP system. The resulting system can also use available logic programming mechanisms in the CPT specification, making it possible to calculate it dynamically, based on the context, rather than by fixed tables. The probabilistic constraint solver uses a Bayesian network internally in order to solve the posed constraints, so this system is also using an underlying propositional Bayesian network for answering queries. Santos Costa et al. indicate [69] as the closest approach to theirs, with the difference that the latter keeps hard constraints on Bayesian variables separate from probabilistic constraints. This allows hard constraints to be solved separately. It is also different in that it does not use conditional independencies (like Bayesian networks do), and therefore its inference is exponential on the number of random variables.

Markov Logic Networks (MLNs) [70] is a recent and rapidly evolving framework for probabilistic logic. Its main distinctions are that it is based on undirected models and has a very simple semantics while keeping the expressive

English / First-Order Logic	Clausal form	Weight
"Smoking causes cancer" $\forall X$ Smokes(X) $\Rightarrow$ Cancer(X)	$\neg$ Smokes(X) $\vee$ Cancer(X)	1.5
"If two people are friends either both smoke or neither does" $\forall X \forall Y$ Fr(X,Y) $\Rightarrow$(Sm(X) $\Leftrightarrow$ Sm(Y))	$\neg$ Friends(X,Y) $\vee$ Smokes(X) $\vee$ Smokes(Y)	1.1

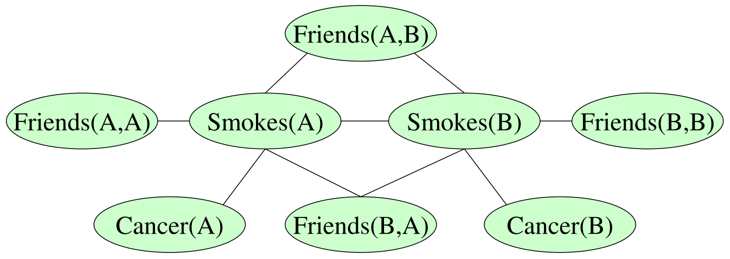


Fig. 12.2. A ground Markov network generated from a Markov Logic Network for objects Anna (A) and Bob (B) (example presented in [70])

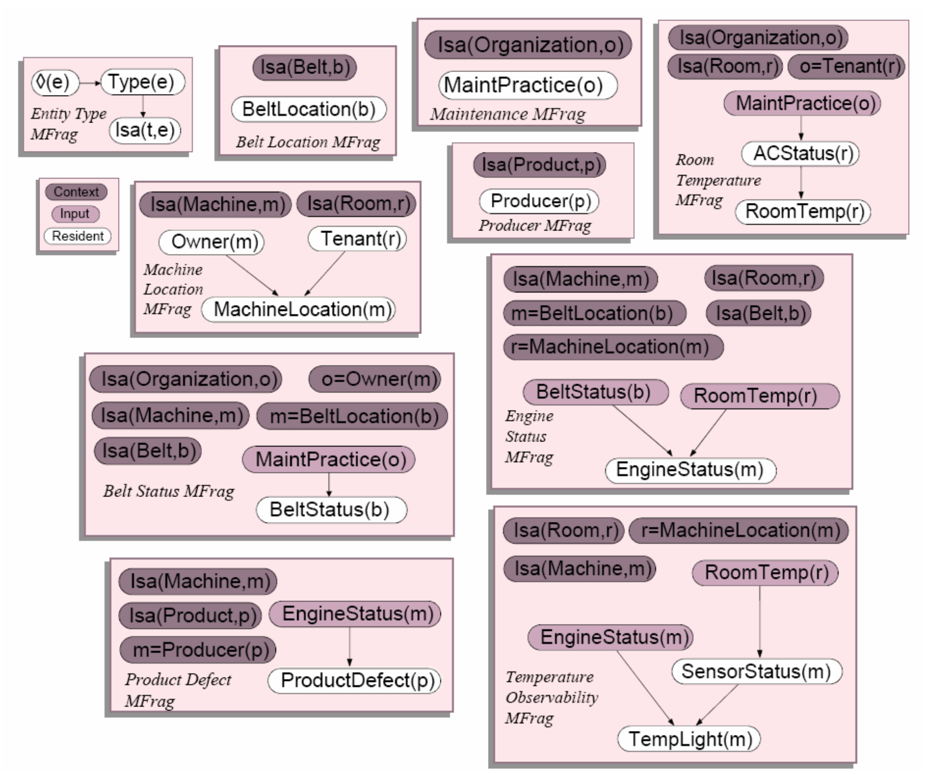


Fig. 12.3. An example of a MEBN, as shown in [72]

power of first-order logic. The downside to this is that its inference can become quite slow if complex constructs are present.

MLNs consist of a set of weighted first-order formulas and a universe of objects. Its semantics is simply that of a Markov network whose features are the instantiations of all these formulas given the universe of objects. The potential of a feature is defined as the exponential of its weight in case it is true. Figure 12.2 shows an example.

Formulas can be arbitrary first-order logic formulas, which are converted to clausal form for inference. Converting existentially quantified formulas to clausal form usually involves Skolemization, which requires uninterpreted functions in the language. Since MLNs do not include such functions, existentially quantified formulas are replaced by the disjunction of their groundings (this is possible because the domain is finite). The great expressivity of MLNs allows them to easily subsume other proposed FOPI languages. They are also a generalization of first-order logic, to which they reduce when weights are infinite.

Learning algorithms for MLNs have been presented from the beginning. Because learning in undirected models is hard, MLNs use the notion of pseudo-likelihood [71], an approximate but efficient method. When data is incomplete, EM is used.

MLNs are a powerful language and framework accompanied by well-supported software (called *Alchemy*) and which has been applied to real domains. The drawback of its expressivity is potentially very large underlying networks (for example, when existential quantification is used).

Laskey [72] presents multi-entity Bayesian networks (MEBNs), a first-order version of Bayesian networks, which rely on generalizing typical Bayesian network representations rather than a logic-like language. A MEBN is a collection of Bayesian network *fragments* involving parameterized random variables. As in the other approaches, the semantics of the model is the Bayesian network resulting from instantiating these fragments. Once they are instantiated, they are put together according to the random variables they share. A MEBN is shown in Fig. 12.3.

Laskey's language is indeed quite rich, allowing infinite models, function symbols and distributions on the parameters of random variables themselves. The work focus on defining this language rather than on the actual implementation, which is based on instantiating a Bayesian network containing the parts relevant to the query at hand. It does not provide a detailed account of this process, which can be especially tricky in the case of infinite models.

12.4.5 Lifted Inference

One of the major difficulties in KBMC approaches is that they must propositionalize the model in order to perform inference. This does not preserve the rich first-order structure present in the original model; the propositionalized version does not indicate anymore that CPTs are instantiations of the same original one, or that random variables are instantiations of an original parameterized random variable. In other words, it creates a potentially large propositional model with a great amount of redundancy that cannot be readily exploited.

Recent research on *lifted inference* [73, 47, 48] has addressed this point. A lifted inference algorithm receives a first-order specification of a probabilistic model and performs inference directly on it, without propositionalization. This can potentially yield an enormous gain in efficiency.

For example, a possible model can be formed by parameterized factors (or *parfactors*) $\phi_1(\mathit{epidemic}(D))$ and $\phi_2(\mathit{sick}(P, D))$ and a set of typed objects $flu, \mathit{rubella}$, and john, mary etc. The model is equivalent to a propositional graphical model formed by all possible instantiations of parfactors by the given objects, which is the set of regular factors $\phi_1(\mathit{epidemic}(flu))$, $\phi_1(\mathit{epidemic}(\mathit{rubella}))$, $\dots$, and $\phi_2(\mathit{sick}(\mathit{john}, flu))$, $\phi_2(\mathit{sick}(\mathit{john}, \mathit{rubella}))$, $\phi_2(\mathit{sick}(\mathit{mary}, flu))$,
$\phi_2(\mathit{sick}(\mathit{mary}, \mathit{rubella}))$, etc.

What lifted inference does, instead of actually generating these instantiations, is to operate *directly* on the parfactors and obtain the *same* answer as the one obtained by instantiating and solving by a propositional algorithm. By operating directly on parfactors, the lifted algorithm can potentially be much more efficient, since the first-order structure is explicitly available to it. For example, suppose we want to compute the marginal of $P(\mathit{epidemic}(flu))$. Then we have to sum out all the other random variables in the model. While a regular KBMC algorithm would instantiate them and then sum them out, a lifted inference algorithm will directly sum out the *parameterized* $epidemic(D)$, for $D \neq flu$, and $sick(P, D)$. The lifted elimination operation may not depend on the number of objects in the domain at all, greatly speeding up the process. The step in which an entire class of random variables is eliminated at once is possible because they all share the same structure, and this structure is explicitly available to the algorithm.

Figure 12.4 presents a simplified diagram of a lifted inference operation.

Poole [73] proposes a lifted algorithm that generalizes propositional Variable Elimination [25] but covers only some specific cases. de Salvo Braz et al. [47, 48] present a broader algorithm, called First-Order Variable Elimination (FOVE). FOVE includes Poole's operation (which this work calls *Inversion Elimination*) a generalized version of it, called simply *Inversion*, and a second elimination operation called *Counting Elimination*. While Inversion does not depend on the domain size, Counting Elimination does, but still only exponentially less than propositionalization. The work also presents rigorous proofs of the correctness of these operations and shows how to solve the lifted version of the Most Probable Explanation (MPE) problem. While more general, FOVE still does not cover all possible cases, when it too must resort to propositionalization. When this

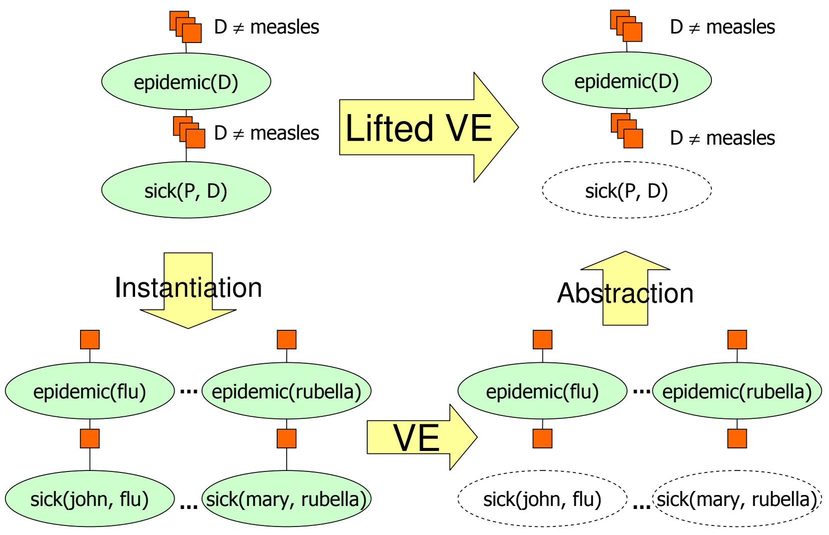


Fig. 12.4. A diagram of several possible operations involving first-order and propositional probabilistic models. The figure uses the notation of factor graphs, which explictly shows potential functions as squares connected to their arguments. Parameterized factors are shown as piled up squares, since they compactly stand for multiple factors. Lifted inference operates solely on the first-order representation and can be much faster than propositional inference, while producing the same results.

happens, this propositionalization will be localized to the parfactors involved in the uncovered case.

Lifted FOPI is a further step towards closing the gap between logic and probabilistic inference, bringing to the latter the type of inference that does not require binding of parameters (which would be the logical variables in atoms, in logical terms), often seen in the former. However, it has its own disadvantages. It is relatively more complicated to implement, and requires a normalizing preprocessing of the model (called *shattering*) that can be very expensive. Further methods are being developed to circumvent these difficulties.

12.5 Relational Learning

In this section we discuss some first-order models developed from a machine learning perspective.

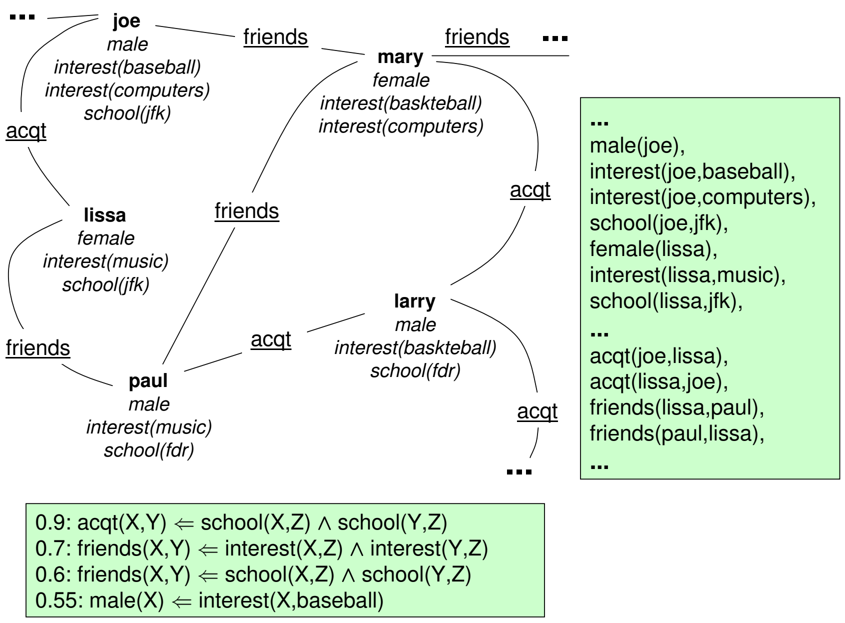
Machine learning algorithms have traditionally been defined as classification of *attribute-value vectors* [74]. In many applications, it is more natural and convenient to represent data as *graphs*, where each vertex represents an object and each edge represents a relation between objects. Vertices can be labeled with attributes of its corresponding object (unary predicates or binary predicates where the second argument is a simple value; this is similar to PRMs in Sect. 12.4.4), and edges can be labeled (the label can be interpreted as a binary predicate holding between the objects). This provides the typical data structure representations such as trees, lists and collections of objects in general. When learning from graphs, we usually want to form hypotheses that explain one or more of the attributes and (or) relations (the *targets*) of objects in terms of its neighbors. Machine learning algorithms which were developed to benefit from this type of representation have often been called *relational.* This is closely associated to probabilistic first-order models, since graph data can be interpreted as a set of ground literals using unary and binary predicates. Because the hypotheses explaining target attributes and relations apply to several objects, it is also convenient to represent the learned hypotheses as quantified (first-order) rules. And because most learners involve probabilities or at least some measure of uncertainty, probabilistic first-order rules provide a natural representation option. Figure 12.5 illustrates these concepts.

We now discuss three forms of relational learning: propositionalization (flattening), Inductive Logic Programming (ILP), and FOPI learning, which can be seen as a synthesis of the two.

12.5.1 Propositionalization

A possible approach to relational machine learning is that of using a relational structure for generating propositional attribute-value vectors for each of its objects. For this reason, the approach has been called *propositionalization.* Because it transforms graph-like data into vector-like data, it is also often called *flattening.*

Cumby & Roth [75] provide a language for transforming relational data into attribute-value vectors. Their concern is not forming a first-order hypothesis,



...
male(joe),
interest(joe,baseball),
interest(joe,computers),
school(joe,jfk),
female(lissa),
interest(lissa,music),
school(lissa,jfk),
...
acqt(joe,lissa),
acqt(lissa,joe),
friends(lissa,paul),
friends(paul,lissa),
...

0.9: acqt(X,Y) ⇐ school(X,Z) ∧ school(Y,Z)
0.7: friends(X,Y) ⇐ interest(X,Z) ∧ interest(Y,Z)
0.6: friends(X,Y) ⇐ school(X,Z) ∧ school(Y,Z)
0.55: male(X) ⇐ interest(X,baseball)

Fig. 12.5. A fragment of a graph structure used as input for relational learning. The same information can be represented as a set of ground literals (right). The hypotheses learned to explain either relations or attributes can be represented as weighted first-order clauses over those literals (below).

however. They instead keep the attribute-value hypothesis and transform novel data to that representation in order to classify it with propositional learners such as Perceptron. For example, in the case of Fig. 12.5, a classifier seeking to learn the relation *acqt* would go through the instances of that predicate and generate suitable attribute-value vectors. The literal $acqt(paul, larry)$ would generate an example with label $acqt(X, Y)$ and features $male(paul)$, $male(larry)$, $interest(paul, music)$, $school(paul, fdr)$, $interest(larry, basketball)$, $school(larry, fdr)$ etc, as well as non-ground ones such as $male(X)$, $male(Y)$, $school(X, fdr)$, $school(Y, fdr)$, $school(X, Z)$, $school(Y, Z)$, $interest(X, music)$ etc. The literal $acqt(joe, lissa)$ would generate an example with label $acqt(X, Y)$ and features $male(joe)$, $female(lissa)$, $interest(joe, baseball)$, $interest(joe, computers)$, $school(joe, jfk)$,etc, as well as non-ground ones such as $male(X)$, $female(Y)$, $school(X, jfk)$, $school(Y, jfk)$, $school(X, Z)$, $school(Y, Z)$ etc. Note how this reveals abstractions – the examples above share the features $school(X, Z)$ and $school(Y, Z)$, which may be one reason for people being acquaintances in this domain. Should a target depend on specific objects (say, it is much more likely for people at the FDR school to be acquainted to each other) not completely abstracted features such as $school(X, fdr)$ would be preferred by the classifier.

There are many different ways of transforming a graph or set of literals into attribute-value vectors for propositional learners. Each of them will represent different learning biases. Some works in this line are LINUS [76], which uses the concept of ij-determinacy (a way of restricting the generalizations of literals) in order to construct hypothesis, 1BC [77] and 1BC2 [78], which differentiate between predicates representing attributes or relations and construct multiset attributes (for example, the set of interests among one's friends), and Relational Bayesian Classifier (RBC) [79], which also uses multiset attribute values with a conditional independence assumption similar to the one used in the Naive Bayes classifier.

One disadvantage of propositionalization is that it performs classification of an object at a time. This prevents the use of possible dependencies between object labels and introduces a bias. These dependencies can be used by FOPI algorithms, which perform *joint* inference over several objects at once. Three of FOPI models which have been specifically developed with this in mind are RDNs [65], RMNs [64] and MLNs [70].

12.5.2 Inductive Logic Programming

Inductive Logic Programming (ILP) stemmed from the logic programming community with the goal of learning logic programs from data rather than writing them. The choice of logic programming as a hypothesis language initially restricted the field to deterministic hypotheses; the later incorporation of probabilities to this framework is one of the origins of FOPI models. However, the fact that these algorithms learn from data give them a statistical flavor even in the deterministic case. For example, Progol [36] uses information-theoretic measures for evaluating deterministic hypotheses.

A typical ILP algorithm works by forming hypotheses one clause at a time. For example, if such an algorithm is trying to learn the concept $mother(X, Y)$, it might generate the clause $mother(X, Y) : -female(X)$, since $female(X)$ does add some predictive power to whether X is a mother, and may at a latter step refine it to $mother(X, Y) : -female(X), child(Y, X)$. This is a top-down approach since the most general clause is successively made more specific according to examples. Two examples of work in this line are [80, 81]. Another approach is bottom-up, best exemplified by Progol [36], where sets of ground literals are successively generalized in order to increase accuracy.

Some ILP algorithms use propositionalization as a subroutine that learns one rule at a time. LINUS [76] transforms its data into attribute-value vectors, applies regular attribute-value learners to it, and then transforms the attribute-value hypothesis into a clause.

Probabilistic ILP (PILP) algorithms (a specific survey can be found at [82]) also often grow clauses and then estimate the probabilities (or parameters) associated with those clauses. Some learn an entire logic program at first, and then the parameters, while others learn the parameters as soon as the clauses are learned. In fact, PILP is simply FOPI with learning done with ILP techniques, and many of these works have been mentioned in previous sections. For example, MLNs [70], PRMs [58, 59] and BLPs [83] learn the structure of their models (as opposed to their parameters) through techniques similar to those of ILP. RDNs [65] use relational classifiers developed in the ILP community as a subroutine to its model learning.

12.6 Restricted First-Order Probabilistic Models

The models presented so far intended to provide a level of expressivity similar to first-order logic. At a minimum, they provide unary and binary predicates arbitrarily applied to a collection of objects. However, there are some probabilistic models exhibiting a restricted subset of first-order aspects, targeted for particular tasks.

One such model is Hidden Markov Models (HMMs) [84], in which a sequence of pairs of random variables represent a *state* and an *observation*. The model relates observations to the state in the same time slice, as well as states in successive time slides. While essentially propositional, HMMs exhibit the sharing of parameters commonly observed in first-order models, since its parameters equally apply to all transitions from one step to the next. The time indices of random variables of an HMM are a restricted form of treating them as first-order.

The same sharing of parameters can be observed in related models. Stochastic Context-Free Grammars [31] consist of set of production rules were a non-terminal symbol is stochastically replaced by a number of possible sequences of symbols. Again, the rules can be applied at different points and parameters are reused. Dynamic Bayesian Networks [85] generalize HMMs in that each step is represented by a full Bayesian network rather than just a pair of state and observations.

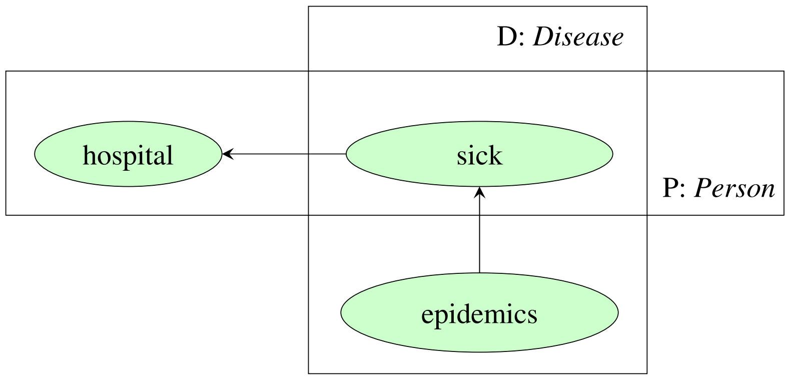


Fig. 12.6. Repeated structure of a graphical model can be indicated by plates. Random variables are implicitly indexed by integer variables associated with the plates inside which they reside. The same CPT is used for all of them.

Other generalizations of these restricted models are Hidden Tree Markov Models [86], where possible states form a tree structure, Logical Hidden Markov Models [66], where each state is represented by a set of logical literals, and Relational Markov Models [87], where each state is also represented by a set of logical literals, but possible arguments follow a taxonomy and induce a lattice on possible literals.

Another simple generalization of propositional models is the plate notation [88], which describes graphical models with parts replicated by a set of indices, indicated by a rectangle involving that part in a diagram. An example is shown in Fig. 12.6. The parameters into these parts are then also replicated. Mjolsness [89] proposes many refinements to this type of notation. The plate notation is commonly used as a tool for descriptions in the literature but has not been typically meant as an input to algorithms.

12.7 Conclusion

First-order probabilistic inference has made much progress in the last twenty years. We believe the main accomplishments have been the clarification the semantics of such languages, as well as a greater understanding on how several different language options relate to each other. In analyzing the works in the area of FOPI, we can distinguish a few main options along which they seem to organize themselves. We now make these aspects more explicit.

12.7.1 Directed vs. Undirected

The decision on using directed or undirected models carries over the the first-order case. While the intelligibility of directed models has favored their use in first-order proposals at first, we can observe a current tendency to use undirected models [70, 90], or at least directed models without the acyclicity requirement [65]. The reason for this shift is that cycles are even more naturally occurring in first-order domains than in propositional ones. A “natural” conditional probability such as $P(smoker(X)|friend(X,Y), smoker(Y))$ creates an underlying network with cycles. The use of undirected models seems to be further justified by the fact that they do not rule out directed models. If the given factors encode conditional probabilities that do not involve cycles, they will still represent the correct distribution even if interpreted as an undirected factor (this however loses structure that could be used to improve efficiency).

12.7.2 The Trade-Off between Language and Algorithm

Some FOPI proposals focus on rich languages that allow the user to indicate domain restrictions which can be exploited for efficiency. Examples of such systems are Ngo and Haddawy’s Probabilistic Logic Programming [23], PRMs [58, 59], LBNs [67]. One example is the treatment of determinism (especially of relations) being expressed explicitly by the language, as in the case of PRMs and LBNs, as discussed in Sect. 12.4.4.

Other solutions propose simple languages, relying on inference algorithms to exploit domain structure (sometimes guided by extra-language directives). The most typical examples are BLPs [66] and MLNs [70].

This choice reflects a trade-off between language and inference algorithm complexities. Complex languages bring built-in optimization hints; for example, PRMs have attributes (with a value) and relations, even though both could be regarded as binary predicates. By indicating that a binary predicate is an attribute, the user implicitly indicates the most efficient ways of using it, which are distinct from the way a relation is used. To obtain the same effect, an algorithm using only a single general notion of binary predicates would have to either find out or be told (by extra-language directives) how to use each of them in the most efficient manner. In this case, compilation and learning can play important roles.

Our particular view is that FOPI languages will tend to become simpler, leaving the complexities to be dealt with by compilation, learning and directives. This reflects the evolution of programming languages, that have increasingly left efficiency details to be dealt with by compilers and directives rather than by the language itself, leaving the latter at a higher level that can be more easily understood and theoretically related to other approaches. On the other hand, this simplicity should not be such as to prevent the development of specialized libraries and knowledge representations. These are useful but best included on top of simple primitives instead of as primitives themselves.

12.7.3 Infinite Models

First-order models may have a finite description while involving an infinite number of random variables, if the number of objects in the domain is infinite. This poses problems to algorithms presented in this survey, since they may not stop in that case. As pointed out by [91], models with infinite number of random variables can be dealt with by approximate, anytime algorithms. These algorithms will provide arbitrarily precise approximations after a sufficient (but finite) amount of computation is performed. Such an approach would also make sense for processing models that, while involving a finite number of random variables, are too complex for exact inference. Such algorithms would also benefit from *guided* evaluation, that is, by choosing to process first the parts of the model that yield greater amounts of information about the query.

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
2. Buchanan, B., Shortliffe, E.: Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project. Addison-Wesley, Reading (1984)
3. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Mateo (Calif.) (1988)

4. Nilsson, N.J.: Probabilistic logic. Artificial Intelligence 28(1), 71–88 (1986)
5. Bacchus, F.: Representing and reasoning with probabilistic knowledge: a logical approach to probabilities. MIT Press, Cambridge (1990)
6. Halpern, J.Y.: An analysis of first-order logics of probability. In: Proceedings of IJCAI 1989, 11th International Joint Conference on Artificial Intelligence, Detroit, US, pp. 1375–1381 (1990)
7. Shortliffe, E.H.: Mycin: a rule-based computer program for advising physicians regarding antimicrobial therapy selection. PhD thesis, Stanford University (1975)
8. Clark, K.L., McCabe, F.G.: Prolog: A language for implementing expert systems. In: Hayes, J.E., Michie, D., Pao, Y.H. (eds.) Machine Intelligence, Ellis Horwood, Chichester, vol. 10, pp. 455–470 (1982)
9. Shapiro, E.: Logic programs with uncertainties: A tool for implementing expert systems. In: Proc. IJCAI 1983, pp. 529–532. William Kaufmann, San Francisco (1983)
10. Heckerman, D.: Probabilistic interpretation for MYCIN's certainty factors. In: Kanal, L.N., Lemmer, J. (eds.) Uncertainty in Artificial Intelligence, pp. 167–196. Kluwer Science Publishers, Dordrecht (1986)
11. Lucas, P.: Certainty-factor-like structures in bayesian belief networks. Knowledge-Based Systems 14, 325–327 (2001)
12. Carnap, R.: The Logical Foundations of Probability. University of Chicago Press, Chicago (1950)
13. Hailperin, T.: Probabilistic logic. Notre Dame Journal of Formal Logic 25(3), 198–212 (1984)
14. Fagin, R., Halpern, J.Y., Megiddo, N.: A logic for reasoning about probabilities. Information and Computation 87(1/2), 78–128 (1990)
15. Fenstad, J.E.: The structure of probabilities defined on first-order languages. Studies in Inductive Logic and Probabilities, pp. 251–262. California Press (1980)
16. Gaifman, H.: Concerning measures in first-order calculi. Israel Journal of Mathematics 2, 1–18 (1964)
17. Gaifman, H., Snir, M.: Probabilities over rich languages, testing and randomness. Journal of Symbolic Logic 47(3), 495–548 (1982)
18. Kifer, M., Li, A.: On the semantics of rule-based expert systems with uncertainty. In: Gyssens, M., Van Gucht, D., Paredaens, J. (eds.) ICDT 1988. LNCS, vol. 326, pp. 102–117. Springer, Heidelberg (1988)
19. Wüthrich, B.: Probabilistic knowledge bases. IEEE Trans. Knowl. Data Eng. 7(5), 691–698 (1995)
20. Ng, R.T., Subrahmanian, V.S.: Probabilistic logic programming. Information and Computation 101(2), 150–201 (1992)
21. Lakshmanan, L.V.S., Sadri, F.: Probabilistic deductive databases. In: Symposium on Logic Programming, pp. 254–268 (1994)
22. Lakshmanan, L.V.S.: An epistemic foundation for logic programming with uncertainty. In: Foundations of Software Technology and Theoretical Computer Science, pp. 89–100 (1994)
23. Ngo, L., Haddawy, P.: Probabilistic logic programming and Bayesian networks. In: Asian Computing Science Conference, pp. 286–300 (1995)
24. Poole, D.: Probabilistic Horn abduction and Bayesian networks. Artificial Intelligence 64(1), 81–129 (1993)
25. Zhang, N.L., Poole, D.: A simple approach to Bayesian network computations. In: Proceedings of the Tenth Biennial Canadian Artificial Intelligence Conference (1994)

26. Lukasiewicz, T.: Probabilistic deduction with conditional constraints over basic events. J. Artif. Intell. Res (JAIR) 10, 199–241 (1999)
27. Frisch, A.M., Haddawy, P.: Anytime deduction for probabilistic logic. Artificial Intelligence 69(1–2), 93–122 (1994)
28. Koller, D., Halpern, J.Y.: Irrelevance and conditioning in first-order probabilistic logic. In: Shrobe, H., Senator, T. (eds.) Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, vol. 2, pp. 569–576. AAAI Press, Menlo Park (1996)
29. Riezler, S.: Probabilistic constraint logic programming (1997)
30. Jaffar, J., Lassez, J.L.: Constraint logic programming. In: POPL 1987: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 111–119. ACM Press, New York (1987)
31. Lari, K., Young, S.: The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language 4, 35–56 (1990)
32. Baker, J.: Trainable grammars for speech recognition. In: Speech communication papers presented at the 97th Meeting of the Acoustical Society, pp. 547–550 (1979)
33. Baum, L.E.: An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. Inequalities 3, 1–8 (1972)
34. Muggleton, S.: Stochastic logic programs. In: De Raedt, L. (ed.) Proceedings of the 5th International Workshop on Inductive Logic Programming, Department of Computer Science, Katholieke Universiteit Leuven, vol. 29 (1995)
35. Cussens, J.: Loglinear models for first-order probabilistic reasoning. In: Laskey, K.B., Prade, H. (eds.) Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI 1999), pp. 126–133. Morgan Kaufmann, San Francisco (1999)
36. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming 13(3-4), 245–286 (1995)
37. Della Pietra, S., Della Pietra, V.J., Lafferty, J.D.: Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(4), 380–393 (1997)
38. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) 39(1), 1–38 (1977)
39. Lukasiewicz, T.: Probabilistic logic programming. In: European Conference on Artificial Intelligence, pp. 388–392 (1998)
40. Baral, C., Gelfond, M., Rushton, J.N.: Probabilistic reasoning with answer sets. In: LPNMR, pp. 21–33 (2004)
41. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge University Press, Cambridge (2000)
42. Sato, T., Kameya, Y.: Prism: A language for symbolic-statistical modeling. In: IJCAI, pp. 1330–1339 (1997)
43. Sato, T., Kameya, Y.: A viterbi-like algorithm and em learning for statistical abduction (2000)
44. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic models with unknown objects. In: Proc. IJCAI (2005)
45. Spiegelhalter, D., Thomas, A., Best, N., Gilks, W.: Bugs: Bayesian inference using gibbs sampling, version 0.30. Technical report, MRC Biostatistics Unit, University of Cambridge (1994)
46. Heckerman, D., Chickering, D.M., Meek, C., Rounthwaite, R., Kadie, C.M.: Dependency networks for inference, collaborative filtering, and data visualization. Journal of Machine Learning Research 1, 49–75 (2000)

47. de Salvo Braz, R.: Lifted First-Order Probabilistic Inference. PhD thesis, University of Illinois at Urbana-Champaign (2007)
48. de Salvo Braz, R., Amir, E., Roth, D.: Lifted first-order probabilistic inference. In: Getoor, L., Taskar, B. (eds.) An Introduction to Statistical Relational Learning, pp. 433–451. MIT Press, Cambridge (2007)
49. Breese, J.S.: Construction of belief and decision networks. Computational Intelligence 8, 624–647 (1991)
50. Wellman, M.P., Breese, J.S., Goldman, R.P.: From knowledge bases to decision models. Knowledge Engineering Review 7, 35–53 (1992)
51. Horsch, M., Poole, D.: A dynamic approach to probabilistic inference using bayesian networks. In: Proceedings of the 6th Conference of Uncertainty in Artificial Intelligence, pp. 155–161. Morgan Kaufmann, San Francisco (1990)
52. Goldman, R.P., Cherniak, E.: A language for construction of belief networks. IEEE Trans. Pattern Anal. Mach. Intell. 15(3), 196–208 (1993)
53. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artificial Intelligence 94(1-2), 7–56 (1997)
54. Glesner, S., Koller, D.: Constructing flexible dynamic belief networks from first-order probalistic knowledge bases. In: Symbolic and Quantitative Approaches to Reasoning and Uncertainty, pp. 217–226 (1995)
55. Haddawy, P.: Generating bayesian networks from probability logic knowledge bases. In: de Mantaras, R.L., Poole, D. (eds.) Uncertainty In Artificial Intelligence 10 (UAI 1994), pp. 262–269. Morgan Kaufmann, San Francisco (1994)
56. Koller, D., Pfeffer, A.: Learning probabilities for noisy first-order rules. In: IJCAI, pp. 1316–1323 (1997)
57. Jaeger, M.: Relational Bayesian networks. In: Kaufmann, M. (ed.) Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, pp. 266–273 (1997)
58. Getoor, L., Friedman, N., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: Džeroski, S., Lavrac, N. (eds.) Relational Data Mining, pp. 307–335. Springer, Heidelberg (2001)
59. Koller, D., Pfeffer, A.: Probabilistic frame-based systems. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI), pp. 580–587 (1998)
60. Minsky, M.: A framework for representing knowledge. In: Computation & intelligence: collected readings, pp. 163–189. American Association for Artificial Intelligence, Menlo Park (1995)
61. Levesque, H.J., Brachman, R.J.: Expressiveness and tractability in knowledge representation and reasoning. Computational Intelligence 3, 78–93 (1987)
62. Koller, D., Levy, A.Y., Pfeffer, A.: P-CLASSIC: A tractable probablistic description logic. In: AAAI/IAAI, pp. 390–397 (1997)
63. Getoor, L.: Learning probabilistic relational models with structural uncertainty. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 322–329. Springer, Heidelberg (2000)
64. Taskar, B., Abbeel, P., Koller, D.: Discriminative probabilistic models for relational data. In: Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI), Edmonton, Canada (2002)
65. Neville, J.: Statistical models and analysis techniques for learning in relational data. PhD thesis, University of Massachusetts Amherst (2006)
66. Kersting, K., De Raedt, L.: Bayesian logic programs. In: Cussens, J., Frisch, A. (eds.) Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming, pp. 138–155 (2000)
67. Fierens, D., Blockeel, H., Bruynooghe, M., Ramon, J.: Logical bayesian networks and their relation to other probabilistic logical models. In: ILP, pp. 121–135 (2005)

68. Santos Costa, V., Page, D., Qazi, M., Cussens, J.: Clp(bn): Constraint logic programming for probabilistic knowledge. In: Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2003), pp. 517–552. Morgan Kaufmann, San Francisco (2003)
69. Angelopoulos, N.: Probabilistic finite domains: A brief overview. In: Stuckey, P.J. (ed.) ICLP 2002. LNCS, vol. 2401, p. 475. Springer, Heidelberg (2002)
70. Richardson, M., Domingos, P.: Markov logic networks. Technical report, Department of Computer Science, University of Washington (2004)
71. Besag, J.: Statistical analysis of non-lattice data. The Statistician 24(3), 179–195 (1975)
72. Laskey, K.B.: First-order Bayesian logic. Technical report, George Mason University Department of Systems Engineering and Operations Research (2005)
73. Poole, D.: First-order probabilistic inference. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 985–991 (2003)
74. Mitchell, T.M.: Machine Learning. McGraw-Hill Higher Education, New York (1997)
75. Cumby, C., Roth, D.: Relational representations that facilitate learning. In: Cohn, A.G., Giunchiglia, F., Selman, B. (eds.) KR 2000: Principles of Knowledge Representation and Reasoning, pp. 425–434. Morgan Kaufmann, San Francisco (2000)
76. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Routledge, New York, 10001 (1993)
77. Flach, P., Lachiche, N.: 1BC: A first-order Bayesian classifier. In: Džeroski, S., Flach, P. (eds.) ILP 1999. LNCS (LNAI), vol. 1634, pp. 92–103. Springer, Heidelberg (1999)
78. Lachiche, N., Flach, P.A.: 1BC2: a true first-order Bayesian classifier. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 133–148. Springer, Heidelberg (2003)
79. Neville, J., Jensen, D., Gallagher, B.: Simple estimators for relational bayesian classifiers. In: ICDM 2003: Proceedings of the Third IEEE International Conference on Data Mining. IEEE Computer Society Press, Washington (2003)
80. Quinlan, J.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)
81. Raedt, L.D., Dehaspe, L.: Clausal discovery. Machine Learning 26(2-3), 99–146 (1997)
82. Raedt, L.D., Kersting, K.: Probabilistic inductive logic programming. In: ALT, pp. 19–36 (2004)
83. Kersting, K., De Raedt, L.: Towards combining inductive logic programming with Bayesian networks. In: Rouveirol, C., Sebag, M. (eds.) ILP 2001. LNCS (LNAI), vol. 2157, pp. 104–117. Springer, Heidelberg (2001)
84. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. In: Readings in speech recognition, pp. 267–296. Morgan Kaufmann Publishers Inc., San Francisco (1990)
85. Murphy, K.P.: Dynamic bayesian networks: representation, inference and learning. PhD thesis, University of California Berkeley, Chair-Stuart Russell (2002)
86. Diligenti, M., Frasconi, P., Gori, M.: Hidden tree markov models for document image classification. IEEE Trans. Pattern Anal. Mach. Intell. 25(4), 519–523 (2003)
87. Anderson, C.R., Domingos, P., Weld, D.S.: Relational Markov models and their application to adaptive web navigation. In: KDD 2002: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 143–152. ACM Press, New York (2002)

88. Buntine, W.L.: Operations for learning with graphical models. Journal of Artificial Intelligence Research 2, 159–225 (1994)
89. Mjolsness, E.: Labeled graph notations for graphical models: Extended report. Technical report, University of California Irvine – Information and Computer Sciences (2004)
90. de Salvo Braz, R., Amir, E., Roth, D.: Lifted first-order probabilistic inference. In: Proceedings of IJCAI 2005, 19th International Joint Conference on Artificial Intelligence (2005)
91. Pfeffer, A., Koller, D.: Semantics and inference for recursive probability models. In: AAAI/IAAI, pp. 538–544 (2000)