In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* Nagoya, Japan, August 1997

Learning probabilities for noisy first-order rules*

Daphne Koller
Stanford University
koller@cs.stanford.edu

Avi Pfeffer
Stanford University
avi@cs.stanford.edu

Abstract

First-order logic is the traditional basis for knowledge representation languages. However, its applicability to many real-world tasks is limited by its inability to represent uncertainty. Bayesian belief networks, on the other hand, are inadequate for complex KR tasks due to the limited expressivity of the underlying (propositional) language. The need to incorporate uncertainty into an expressive language has led to a resurgence of work on *first-order probabilistic logic*. This paper addresses one of the main objections to the incorporation of probabilities into the language: "Where do the numbers come from?" We present an approach that takes a knowledge base in an expressive rule-based first-order language, and learns the probabilistic parameters associated with those rules from data cases. Our approach, which is based on algorithms for learning in traditional Bayesian networks, can handle data cases where many of the relevant aspects of the situation are unobserved. It is also capable of utilizing a rich variety of data cases, including instances with varying causal structure, and even involving a varying number of individuals. These features allow the approach to be used for a wide range of tasks, such as learning genetic propagation models or learning first-order STRIPS planning operators with uncertain effects.

1 Introduction

First-order logic has traditionally formed the basis for most large-scale knowledge representation systems. The advantages of first-order logic in this context are obvious: The notions of "individuals", their properties, and the relations between them provide an elegant and expressive framework for reasoning about many diverse domains. The use of quantification allows us to compactly represent general rules, that can be applied in many different situations. For example, when reasoning about genetic transmission of certain properties (e.g., genetically transmitted diseases), we can write down general rules that hold for all people and many properties.

Unfortunately, like all deterministic logics, first-order logic is highly limited in its ability to represent our uncertainty about the world. A fact is either known to be true, known to be false, or neither. One cannot say that a fact is probably true. Real-world relationships, on the other hand, are noisy and nondeterministic, a fact which cannot be captured in the standard logical framework. This severely limits the applicability of this framework. For example, very few deterministic rules in the domain of genetically transmitted properties are actually (absolutely) true in real life.

This limitation, which is crucial in many domains (e.g., medical diagnosis), has led over the last decade to the resurgence of probabilistic reasoning in AI. In particular, Bayesian belief networks [Pearl, 1988] have been shown to be a principled and useful framework for reasoning in an uncertain domain. However, belief networks do not, by themselves, provide a complete solution for very large-scale knowledge representation tasks. The primary reason is their attribute-based (propositional) nature, which does not support a domain description in terms of general rules that apply to many qualitatively different situations.

The tension between these two complementary paradigms has been the primary motivation for some of the recent work on trying to combine the two [Halpern, 1990; Breese, 1992; Poole, 1993; Ngo *et al.*, 1995]. *Knowledge-based model construction (KBMC)* goes a considerable way towards bridging this gap by allowing a set of *first-order probabilistic logic (FOPL)* rules (first-order rules with associated probabilistic uncertainty parameters) to be used as a basis for generating Bayesian networks tailored to particular problem instances.

The idea of attaching probabilistic parameters to rules leaves unanswered one of the major objections that have been raised about probabilistic representations: the famous "where do the numbers come from" question. This issue has been addressed satisfactorily for traditional belief networks [Lauritzen, 1995; Heckerman, 1995]. In this paper, we show how similar techniques can be used to learn the probabilistic parameters of FOPL rules from data.

The ability to learn the uncertainty parameters of a rich first-order representation has the potential to be a powerful tool in many situations. We illustrate this using two very different examples: reasoning about genetically transmitted properties and planning for mobile robots. In both these examples, as in many others, the problem of learning these uncertainty parameters is an important one. The skeleton of the rules (the rules without the parameters) is often easier to acquire than

*This work was supported in part through the generosity of the Powell foundation, and by ONR grant N00014-96-1-0718.

the parameters themselves. In fact, an existing set of traditional first-order rules often provides us with an appropriate skeleton. In general, the rule structure directly reflects the underlying causal structure of the domain, a type of knowledge with which human experts are often fairly comfortable. By contrast, probabilistic parameters are notoriously difficult to elicit from people.

The propagation of genetically transmitted properties was a key example in some of the early research into belief networks [Lauritzen and Spiegelhalter, 1988]. Given a particular family tree and set of properties being studied, one can construct a traditional propositional belief network in which the probability of each property being passed from each generation to the next is represented. However, a new network must be specially constructed for every family tree. Currently, this is either done manually or using a special-purpose procedural program [Szolovits and Pauker, 1992].

Using a first order representation, one can capture the general mechanism of gene inheritance using a small number of rules. These can be used to automatically generate an appropriate belief network for any family tree and any set of properties. The generality of first-order languages is manifested here in three ways: the same mechanism is at work in different family trees, in different generations within the same family tree, and in the propagation of different genes from parent to child. Our approach enables us to learn the propagation parameters for the different classes of properties. We will also be able to learn the strength of the correlations between propagations of the different properties, e.g., that eye color is usually propagated together with diabetes (because of proximity on the chromosome strand).

The task of planning for an autonomous agent has traditionally been based on a logical representation of actions and their effects. In recent years, there has been a growing consensus that the underlying assumptions of this representation—deterministic actions, reliable sensors, and (often) complete observability—are rarely true in practice, particularly in robotics applications. As a consequence, probabilistic representations have recently started to play a role in planning [Kushmerick *et al.*, 1993; Dean and Wellman, 1991]. These representations, however, are typically attribute-based, and are therefore limited in their ability to capture general patterns in action models. FOPL would allow for an integration of these two formalisms.

Certain important issues arise when we contemplate learning in domains such as these. First, as in most real-life applications, most of the relevant variables are not observed by the learning agent. In the domain of genetically transmitted properties, we may observe the phenotype of some of the people in the family. We will rarely (if ever) have complete information about the entire family. We will hardly ever have any information at all about the *genotype* of the different people involved. In the planning domain, we will typically have access only to the robot's sensors. The variables corresponding to the true state of the world are almost always unobservable.

A second common thread is that these domains all require the ability to learn from data cases that are qualitatively very different from each other. For example, in the genetic propagation example, we wish to learn from different family trees, over a varying number of individuals, and representing the inheritance of different properties. In our planning domain, we are faced with runs of different length, where the robot undertakes different actions. Note that, in both of these applications, each of the (very different) data cases gives us information about many (or all) of the parameters of interest, so that we cannot simply separate them into distinct clusters and run a learning algorithm on each.

The approach we present in the paper is capable of dealing with both of these issues. We start out with a knowledge base consisting of partially specified FOPL rules: the rule structure is determined, but the uncertainty parameters are left unspecified. We are also given a set of data cases, where each data case consists of a context (e.g., the structure of the family tree, or the set of actions taken by the robot) and the observations made by the agent. We use a standard KBMC algorithm to generate the network structure for each of the data cases. The observations in each data case become evidence in the resulting network. The conditional probability tables in the resulting networks are related to the parameters corresponding to the rules in the knowledge base. We adaptively learn these parameters, using an extension to the standard EM algorithm [Lauritzen, 1995] for learning the parameters of a belief network with fixed structure and hidden variables. We extend it to deal with an ensemble of networks of varying structure, and in which the same parameter can appear several times.

The two major advantages of first order languages over propositional languages—generality and compactness—have particular ramifications in a learning context. The generality of first-order models allows the learned parameters to be reused again and again in many different contexts. The cost of learning is therefore amortized over a large number of instances in which the benefits are reaped. The compactness of such representations allows a probabilistic model to be represented using a small number of parameters, hopefully resulting in faster learning.

2 Knowledge-based model construction

Since the idea of constructing belief networks from a first-order probabilistic knowledge base was first proposed [Breese, 1992], several approaches have been developed. Most of these augment logic-programming style rules with uncertainty parameters. In this paper, we largely follow the framework of [Ngo *et al.*, 1995]. In this approach, a set of Horn rules describes the ways in which first-order predicates influence each other. Because the influence may be uncertain, each rule has a *uncertainty parameter* associated with it. Intuitively, one can think of a rule as identifying a possible set of conditions under which the consquence becomes true, and giving the probability that the consequence actually becomes true as a result of the conditions.

For example, a very simple model for gene propagation can be expressed in the rules:

$$\text{genotype(P,G)} \overset{0.5}{\longleftarrow} \text{parent(P,Q), genotype(Q,G).}$$
$$\text{phenotype(P,G)} \overset{0.75}{\longleftarrow} \text{genotype(P,G).}$$

The first rule says that a when a person's parent has a gene, the person will inherit it with probability 0.5. The second rule says that when a person has a gene, it will be observed in the person's phenotype with probability 0.75.

When only one instantiation of a rule can cause a predicate to be true, the associated uncertainty parameter is in fact the conditional probability that the head is true given that the body is true. Sometimes, more than one set of conditions can cause a predicate to be true. For example, if both a person's parents have a gene, the first rule will fire twice, for the two different values of Q. In such cases, we need a *combination rule* to indicate how the different possible causes interact. One very common combination rule is *noisy-or* [Pearl, 1988], which describes a situation in which an effect happens whenever any of its potential causes succeeds in making it happen, and the different causal influences act independently. More precisely, the probability that the effect does not happen is the probability that all the potential causes independently fail to cause it. For example, if the combination rule for genotype is noisy-or, then a person both of whose parents have a gene will fail to inherit it with probability $(0.5)^2 = 0.25$.

A more accurate model for genetic propagation may incorporate the number of chromosomes (0, 1, or 2) on which the gene is found. (Our language allows for multi-valued variables. Rules for such variables have several parameters corresponding to each of the possible values.) A person with the gene on both chromosomes in a pair will always propagate a copy to his or her children, while a person with one copy will only propagate it with probability 0.5. The combination rule in this case will be a noisy addition rule in which the number of genes possessed by a person is the sum of the number of succesful propagations from his or her parents. The probability that a gene will be manifested in a person's phenotype will depend both on the number of copies possessed and on whether the gene is dominant or recessive. Note that this can easily be expressed in our language as a property of G. An even richer model may consider the correlation between the propagation of different genes based on their proximity on the chromosome strand.

The rules in the knowledge base describe, in a general manner, the ways in which various predicates interact. They are used in a particular situation to build a Bayesian network, via a process of *knowledge-based model construction* (KBMC). The resulting network defines a probability distribution over the variables that are relevant in the given situation. A situation is defined by a *context*, which determines the structural relationship between the objects in the situation. In the genetic domain, the parent predicate is part of the context, defining the family tree. In general, the body of a rule will consist of both context variables and random variables, which are treated differently by the model construction process.

The KBMC algorithm takes as input a knowledge base, a context, a query and evidence, and returns a Bayesian network that can be used to compute the probability of the query given the evidence. The context, the query, and the evidence are all ground facts in the language. The algorithm proceeds by backward chaining through the Horn rules, iteratively adding nodes representing different ground facts to the network. It starts by adding the query and the evidence. Each time a variable is added to the network, it is matched with the rule heads to determine what predicates can influence it. Context predicates appearing in the rule body must be satisfied for the rule to apply; if it does, the other predicates in the body (appropriately instantiated) are added to the network as random variables. Figure 1 shows a simple network constructed for the genetic domain to compute the probability of phenotype(a, big_ears) given phenotype(b, big_ears) and not phenotype(c, big_ears). The context consists of the ground facts parent(a,c), parent(b,c), parent(a,d), and parent(b,d).

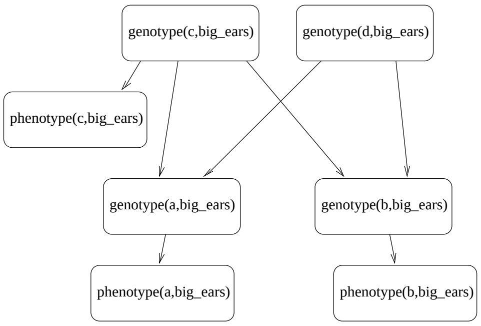


Figure 1: Generated network for genetic domain.

In order to complete the specification of the probability distribution defined by the Bayesian network, the KBMC algorithm must determine the conditional probability table (CPT) for each node. This table lists the conditional probability of the node given each possible value of its parents. The CPT entries are determined by the uncertainty parameters, using the combination rules. In principle, the combination rules could determine the entries to be any function of the parameters. However, learning is greatly simplified if each CPT entry is associated with at most one parameter which must be learned. Thus, we restrict attention to *decomposable* combination rules, ones which can be expressed using a set of separate nodes corresponding to the different influences, which are then combined in another node. Fortunately, all the commonly occuring combination rules (including noisy-or and tree-structured [Boutilier *et al.*, 1996]) generate CPTs with this property. The KBMC algorithm automatically generates the decomposed representation for these combination rules, thereby facilitating learning.

This approach can be applied naturally to planning domains. For example, consider planning in a robotics domain where properties of objects and the effects of actions such as moving and grasping are uncertain. For any (possibly uncertain) initial condition and sequence of actions, KBMC is used to build a probabilistic model of the world after the actions have been taken. The context here consists of the set of objects in the world, some known properties of the objects, such as their type and shape, and the sequence of actions taken. The random variables include other properties of the objects (such as whether they are currently wet and therefore harder to grab), and the locations of objects at different times. The knowledge base consists of STRIPS-like rules with uncertainty (as in [Kushmerick *et al.*, 1993]), stipulating the probability that certain postconditions will hold given that the preconditions hold and an action is taken. Our use of the closed world assumption on context predicates fits naturally with the STRIPS assumptions.

A useful type of combination rule for planning domains is a *selection rule*. A selection rule behaves analogously to a multiplexer: a predicate may match several rules, and the value of a selection variable determines which of the rules is applicable. Selection rules are useful in situations where a predicate is influenced by a single cause, but the identity of the cause is itself uncertain. This is typical of planning situations. For example, the effects of a move action may depend on the properties of the robot's current location. Since the robot's location is itself a random variable, the properties of all locations could potentially influence the action's effects. The location is used as a selector to determine which properties are the relevant ones. Computationally, selection rules require the Bayesian network inference algorithm to take advantage of *context-specific independence* [Boutilier *et al.*, 1996].

3 Learning

Our learning task is to take a set of data cases, $\mathcal{C}$, and return a hypothesis H that "explains" the data $\mathcal{C}$ in the best possible way. The hope is that a hypothesis that provides a good explanation will also generalize well to unseen data cases (modulo concerns about overfitting). Here, each data case consists of a context and some evidence. Since our goal is to learn the parameters for a set of FOPL rules, we assume that our algorithm is provided with a skeleton rule base, and must only fill in the values of the uncertainty parameters. Formally, our hypothesis space consists of the possible values for the rule parameters. We assume for convenience that the rule parameters have values between 0 and 1. Thus, if there are M unspecified parameters, the hypothesis space is the set of M-vectors $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_M) \in [0,1]^M$. We consider a hypothesis $\boldsymbol{\lambda}$ as a good explanation for the data if it gives it high probability. Thus, we seek to find the *maximum likelihood* hypothesis $\boldsymbol{\lambda}$ that maximizes the probability of the data $\mathcal{C}$.

Our first task is to define this probability $\Pr_\lambda(\mathcal{C})$. For a single data case $C \in \mathcal{C}$, we can use the techniques of Section 2. A probabilistic model consisting of the rule skeletons and the parameters $\boldsymbol{\lambda}$ defines a belief network for each data case. The probability of the data case given the hypothesis is thus defined as the probability that the evidence variables will take on their given values in the distribution defined by this belief network. More precisely, let C be a particular data case, $\boldsymbol{D}^C$ the observed evidence in the data case, and $\mathcal{N}^C_\lambda$ the belief network constructed for that data case from its context. The probability of C given the hypothesis $\boldsymbol{\lambda}$ is defined as $\Pr_\lambda(C) \stackrel{\text{def}}{=} \Pr_{\mathcal{N}^C_\lambda}(\boldsymbol{D}^C)$.

At this point we would like to define the likelihood of the entire data set as the product of the likelihoods of the individual data cases, so that if $\mathcal{C} = \{C_1, \ldots, C_N\}$, then $\Pr_\lambda(\mathcal{C}) = \prod_{i=1}^k \Pr_\lambda(C_i)$. This definition embodies the assumption that the different data cases are independent. This seemingly innocuous assumption, which is made almost universally in the context of machine learning, is not as obviously justified. Our more expressive language gives us the ability to relate two individuals, having the properties of one affect the other. But now, we may be uncertain about whether individuals observed in the context of two different data cases are related to each other. In the gene inheritance domain, for example, two people appearing in different family trees may in fact have a common ancestor, thereby linking the two trees. In some sense, the 'ideal' model for a data set $\mathcal{C}$ is a single huge belief network incorporating all of our information. After all, our domain really does contain all of these elements. In this network, we can represent our uncertainty concerning the potential relationships between individuals in the different family trees. Clearly, practical considerations prevent us from taking this course. This is the reason for making the independence assumption, which in the genetics example essentially asserts that different family trees are extremely unlikely to be closely related to each other, and that the influences between data cases are attenuated across many generations. However, it is important to keep in mind that this is purely an approximation, and that we need to check every time whether it is justified in our particular situation.

4 The Learning Algorithm

In this section we describe our learning algorithm in detail. The algorithm takes as input a set of probabilistic rule skeletons with some of the rule parameters left unspecified, and a training set $\mathcal{C}$ consisting of contexts and evidence. It attempts to find the maximum likelihood vector of parameter values using a two-stage process. In the first stage, it constructs a belief network for each data case by mimicking the knowledge based model construction process. In the second stage, it attempts to find the maximum likelihood hypothesis using the EM method (in a manner analogous to its use for learning Bayesian networks [Lauritzen, 1995]).

Let C be a particular data case, $\boldsymbol{D}^C$ the evidence in the data case, and $\mathcal{N}^C$ the the belief network constructed for that data case. The learning algorithm begins by building the network for each data case, via back-chaining from the evidence nodes. The second phase of the algorithm uses the EM algorithm to search for the value of $\boldsymbol{\lambda}$ that maximizes the likelihood of the evidence in the constructed belief networks. We briefly review the EM algorithm and its application to our problem. To understand the intuition, consider the problem of maximum likelihood parameter learning in standard Bayesian networks from fully observable data. There, the network structure is identical in all data cases, and the parameters are simply the CPT entries. Let X be some node in the network and $\boldsymbol{U}$ be its parents. The maximum likelihood estimate for the CPT entry $\Pr(X = x \mid \boldsymbol{U} = \boldsymbol{u})$ is simply the number of data cases where $X, \boldsymbol{U}$ take the values $x, \boldsymbol{u}$ respectively, divided by the number of data cases where $\boldsymbol{U}$ takes the value $\boldsymbol{u}$.

If our data cases have missing values, we can no longer perform this counting process. The EM algorithm essentially provides us with a way for probabilistically "filling in" the missing values. It starts out with some initial set of parameters, and uses them to compute a probability distribution over the various possible completions of each partial data case. Each completion is then treated as a fully-observed data case, but one whose weight is its probability. A new set of parameters is then computed as described above, over the set of weighted data cases. The process is now repeated with the new set of parameters. Standard results (see [McLachlan and Krishnan, 1997]) imply that this procedure converges to a set of parameters which is a local maximum in the likelihood space. In practice, of course, one cannot generate every fully observable completion for a partially observable data case C, since the number of such completions is exponential in the number of unobserved variables in C. Luckily, the total weight of the completions for C which contribute to the weighted count of the event $X = x, \boldsymbol{U} = \boldsymbol{u}$ is simply $\Pr(X = x, \boldsymbol{U} = \boldsymbol{u} \mid \boldsymbol{D}^C)$.

In our context, the basic idea is the same. The two main differences are that the networks for the different data cases have different structures, so that a parameter may appear in a variety of contexts, and that the same parameter can appear more than once in the same network. To see that neither of these is a problem, consider some rule r in our knowledge base. By assumption, r is associated with some set of parameters, and these appear only in r. Recall that our use of decomposable combination rules implies that each node in the generated network is associated with at most one rule. (Some nodes simply compute deterministic functions such as *or* and summation.) Thus, while the same rule can induce more than one node in the network for a data case, all of the nodes have an identical local structure: the CPTs are the same and incorporate the parameters in the same way. Thus, each of the nodes derived from r can be viewed as a separate "experiment" for the parameters associated with r. Each should therefore make a separate contribution to the "count" for those parameters. This situation is now analogous to the one that arises in learning *Hidden Markov Models* [Rabiner and Juang, 1986], where we also have data cases of varying structure and parameter sharing in each data case.

Formally, let r be some rule, and λ be one of its parameters. Let $x_r, \boldsymbol{u}_r$ be the values for a node and its parents that are associated with λ in a node generated by rule r. For every data case C, let $\mathcal{X}_r^C$ be the set of nodes in the network $\mathcal{N}^C$ which correspond to the rule r. In each iteration of the EM algorithm, we begin with some set of parameters $\boldsymbol{\lambda}$, and adjust each of them according to the weighted counts, as follows:

$$\textit{new-}\lambda \leftarrow \frac{\sum_C \sum_{X \in \mathcal{X}_r^C} \Pr_\lambda(X = x, \text{Parents}(X) = \boldsymbol{u} \mid \boldsymbol{D}^C)}{\sum_C \sum_{X \in \mathcal{X}_r^C} \Pr_\lambda(\text{Parents}(X) = \boldsymbol{u} \mid \boldsymbol{D}^C)}$$

The values of $\Pr_\lambda$ can be computed using standard Bayesian network calculations in the network $\mathcal{N}^C$, constructed using the current guess for $\boldsymbol{\lambda}$.

The fact that the we get the maximum likelihood estimate for our parameters in the fully observable case implies the desired convergence property [McLachlan and Krishnan, 1997]:

Theorem 1: *The iterative EM procedure described above converges to a set of parameters $\boldsymbol{\lambda}$ which induces a local maximum in the likelihood* $\Pr_\lambda(\mathcal{C})$.

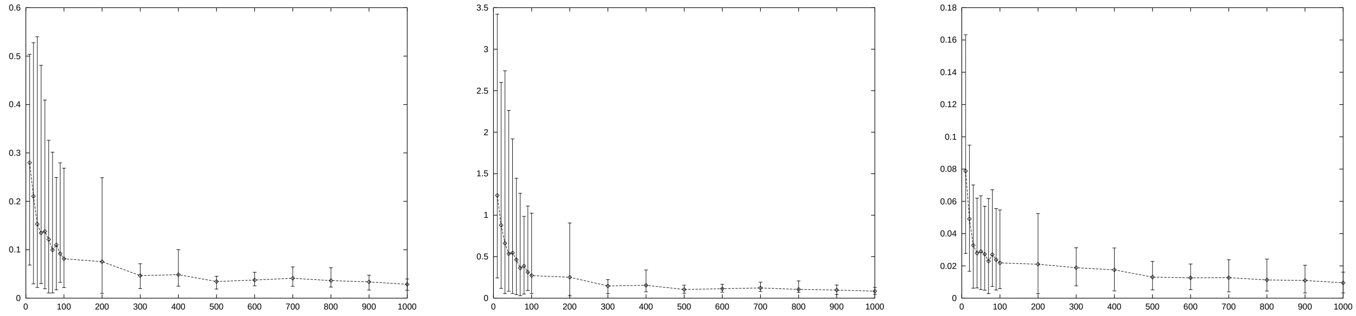
It is instructive to compare the complexity of our learning algorithm to that of parameter estimation in standard (propositional) Bayesian networks. Our learning procedure involves a single initial phase of constructing the network structure for each data case. The cost of this phase is insignificant relative to the cost of the EM iterations, each of which involves running Bayesian network inference for each data case. This cost (per iteration) is the same as in the case of learning parameters for propositional Bayesian networks. We do not have an analysis of the number of iterations required for convergence either for standard Bayesian networks or for our rules. The first-order rules, however, support a significant reduction in the dimensionality of the parameter space via the parameter sharing encoded in the rules. In general, a reduction in the number of parameters tends to speed up convergence. For example, it has recently been shown [Friedman and Goldszmidt, 1996] that exploiting context-specific independence in traditional Bayesian networks can speed up the learning process considerably.

The learning procedure presented here suffers from two potential problems: local maxima and overfitting. Since we are only attempting to learn numerical parameters, any overfitting would be numerical, i.e., learning the parameter values to too great a degree of accuracy. Techniques such as random restart may alleviate the problem of local maxima, but possibly at the cost of increasing the danger of overfitting. Future work should determine how serious these issues are for this procedure, and develop techniques to deal with them.

5 Experimental Results

We tested the learning algorithm on a simple gene propagation model with three parameters. We generated data cases from a given set of rules with associated parameters. We then gave our algorithm the "correct" rule structures and used it to learn the parameters from the data cases. In addition to the two rules shown in Section 2, there was a rule for spontaneous acquisition of a gene, with uncertainty parameter 0.05. The experiments tested the ability of the algorithm to learn the correct values of the parameters from n data cases, for various values of n between 10 and 1000. Each data case described a family tree relating between 20 and 40 people, with the phenotype being observed for approximately one third of them. Ten sets of experiments were run for every value of n, each with a different training set constructed from the same set of parameters.

The results are shown in Figure 2. Figure 2(a) shows the mean absolute error of the learned parameter values as compared to their true values. The graph shows the average, best and worst results for each value of n. Figure 2(b) shows the relative error for the parameter values. Figure 2(c) describes the performance of the learned parameters in predicting the probabilities of events in a test set. The test set consisted of 500 data cases, generated from the same model as the training data, but which were not shown to the learning algorithm. The figure shows the mean relative error of the predicted probabilities as compared to the true probabilities. Notice that the relative error of the predictions is much smaller than the relative error of the parameter values. (Note the scale of the two graphs.) It has often been observed that the predictive performance of a Bayesian network is not sensitive to small error in the parameters. Our results indicate that a similar phenomenon may hold for the parameters of noisy rules. This type of robustness to small errors greatly increases the appli-



(a) (b) (c)

Figure 2: (a) Mean absolute error of learned parameter values. (b) Mean relative error of learned parameter values. (c) Relative error on probability prediction for new cases. In each figure, the x-axis represents the number of data cases.

cability of learning.

6 Conclusion

We have shown how techniques for learning in standard belief networks can be adapted to learning in first-order probabilistic models. This allows us to learn in domains where we encounter many qualitatively different circumstances that share an underlying causal structure and uncertainty parameters. Clearly, more extensive experiments are required in order to test the usefulness of our approach in practice.

Our presentation in this paper was based on a specific representation language for the first-order probabilistic rules. We are currently investigating the problem of defining expressive languages for modeling complex stochastic domains, including languages that support the representation of continuous variables and temporal processes, and reasoning at different levels of granularity. Whatever the results of this research, we expect that a model in our language will continue to define a belief network for a given situation, and that the conditional probabilities will be functions of various parameters. Therefore, the ideas in this paper should continue to be applicable.

Finally, we have focused on learning the numeric uncertainty parameters of first-order probabilistic rules. We did not address the problem of learning the structure of the rules. In recent years, there has been significant work both on learning the structure of belief networks (see [Heckerman, 1995] for a survey) and on *inductive logic programming* [Muggleton, 1992]—learning deterministic first-order rules. It would be very interesting to see whether the techniques developed in these two areas of research can be integrated, allowing us to learn the causal/rule structure of complex uncertain domains.

References

[Boutilier *et al.*, 1996] C.E. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in bayesian networks. In *Proc. UAI*, 1996.

[Breese, 1992] J.S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 1992.

[Dean and Wellman, 1991] T. Dean and M. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[Friedman and Goldszmidt, 1996] N. Friedman and M. Goldszmidt. Learning bayesian networks with local structure. In *Proc. UAI*, 1996.

[Halpern, 1990] J. Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46, 1990.

[Heckerman, 1995] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[Kushmerick *et al.*, 1993] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proc. IJCAI*, 1993.

[Lauritzen and Spiegelhalter, 1988] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, B 50(2), 1988.

[Lauritzen, 1995] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19, 1995.

[McLachlan and Krishnan, 1997] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley Interscience, 1997.

[Muggleton, 1992] S. Muggleton. *Inductive Logic Programming*. Academic Press, 1992.

[Ngo *et al.*, 1995] L. Ngo, P. Haddawy, and J. Helwig. A theoretical framework for context-sensitive temporal probability model construction with application to plan projection. In *Proc. UAI*, 1995.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[Poole, 1993] David Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1), 1993.

[Rabiner and Juang, 1986] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 1986.

[Szolovits and Pauker, 1992] P. Szolovits and S.P. Pauker. Pedigree analysis for genetic counseling. In *Proc. 7th World Congress on Medical Informatics*, 1992.