Learning structure and parameters of Stochastic Logic Programs*

S. Muggleton,
Department of Computing,
Imperial College,
London,
United Kingdom.

Abstract

Previous papers have studied learning of Stochastic Logic Programs (SLPs) either as a purely parametric estimation problem or separated structure learning and parameter estimation into separate phases. In this paper we consider ways in which both the structure and the parameters of an SLP can be learned simultaneously. The paper assumes an ILP algorithm, such as Progol or FOIL, in which clauses are constructed independently. We derive analytical and numerical methods for efficient computation of the optimal probability parameters for a single clause choice within such a search. We then describe how this technique has been incorporated into Progol4.5.

Keywords: Stochastic logic programs, generalisation, analytical methods, numerical methods.

1 Introduction

Stochastic Logic Programs (SLPs) [11] were introduced originally as a way of lifting stochastic grammars to the level of first-order Logic Programs (LPs). Later Cussens [1] showed that SLPs can be used to represent undirected Bayes' nets. SLPs have been used [9] to define distributions for sampling within Inductive Logic Programming (ILP) [7].

Previous papers have studied learning of Stochastic Logic Programs (SLPs) either as a purely parametric estimation problem [2] or separated structure learning and parameter estimation into separate phases [12]. In this paper we consider ways in which both the structure and the parameters of an SLP can be

*This paper extends one of the same name presented at ILP02 by introduction of a new analysis to the n example case and the description of an implementation in Progol4.5.

learned simultaneously. We assume an ILP algorithm, such as Progol [6] or FOIL [14], in which clauses are constructed independently. Analytical and numerical methods are derived for efficient computation of the optimal probability label for a single clause choice within such a search. The search is implemented in Progol4.5.

The paper is arranged as follows. Section 2 gives a general introduction to SLPs. The generalisation model for SLPs described in [12] is reviewed in Section 3. Section 4 describes the problem of choosing the probability label for a single new clause, assuming all other clauses and probability labels in the SLP remain fixed. Equations for the general case are derived using calculus in Section 4.1. A closed form solution for the case in which exactly two examples are present is described in Section 4.2. Iterative numerical methods for solving the general case are described in Section 4.3. Section 5 provides an overview of Progol. It is then described in Section 6 how user-defined evaluation functions in Progol4.5 allow the implementation of the approach described in this paper. Related work is then discussed in Section 7. Section 8 describes provides conclusions and a description of further work.

2 Stochastic logic programs

2.1 Syntax of SLPs

An SLP S is a set of labelled clauses $p{:}C$ where p is a probability (ie. a number in the range $[0,1]$) and C is a first-order range-restricted definite clause[1]. The subset S_p of clauses in S with predicate symbol p in the head is called the definition of p. For each definition S_p the sum of probability labels π_p must be at most 1. S is said to be complete if $\pi_p = 1$ for each p and incomplete otherwise. $P(S)$ represents the definite program consisting of all the clauses in S, with labels removed.

Example 1 Unbiased coin. *The following SLP is complete and represents a coin which comes up either heads or tails with probability 0.5.*

$$S_1 = \left\{ \begin{array}{l} 0.5 : coin(head) \leftarrow \\ 0.5 : coin(tail) \leftarrow \end{array} \right\}$$

S_1 is a simple example of a sampling distribution

Example 2 Pet example. *The following SLP is incomplete.*

$$S_2 = \left\{ \begin{array}{ll} 0.3 : likes(X,Y) \leftarrow & pet(Y,X), pet(Z,X), \\ & cat(Y), mouse(Z) \end{array} \right\}$$

S_2 shows how statements of the form $\Pr(P(\vec{x})|Q(\vec{y})) = p$ can be encoded within an SLP, in this case $\Pr(\text{likes(X,Y)}|\ldots)) = 0.3$.

[1] Cussens [1] considers a less restricted definition of SLPs.

2.2 Proof for SLPs

A Stochastic SLD (SSLD) refutation is a sequence $D_{S,G} = \langle 1{:}G, p_1{:}C_1, .., p_n{:}C_n\rangle$ in which G is a goal, each $p_i{:}C_i \in S$ and $D_{P(S),G} = \langle G, C_1, .., C_n\rangle$ is an SLD refutation from $P(S)$. SSLD refutation represents the repeated application of the SSLD inference rule. This takes a goal $p{:}G$ and a labelled clause $q{:}C$ and produces the labelled goal $pq{:}R$, where R is the SLD resolvent of G and C. The answer probability of $D_{S,G}$ is $Q(D_{S,G}) = \prod_{i=1}^{n} p_i$. The incomplete probability of any ground atom a with respect to S is $Q(a|S) = \sum_{D_{S,(\leftarrow a)}} Q(D_{S,(\leftarrow a)})$. We can state this as $S \vdash_{SSLD} Q(a|S) \leq Pr(a|S) \leq 1$, where $Pr(a|S)$ represents the conditional probability of a given S.

Remark 3 Incomplete probabilities. *If a is a ground atom with predicate symbol p and the definition S_p in SLP S is incomplete then $Q(a|S) \leq \pi_p$.*
Proof. *Suppose the probability labels on clauses in S_p are $p_1, .., p_n$ then $Q(a|S) = p_1 q_1 + .. + p_n q_n$ where each q_i is a sum of products for which $0 \leq q_i \leq 1$. Thus $Q(a|S) \leq p_1 + .. + p_n = \pi_p$.*

3 Generalisation model

We assume an ILP framework in which we are provided with a background SLP S which corresponds to the underlying logic program $B = P(S)$. It is further assumed that S is complete. In addition we are given a set of ground unit positive examples E. The aim is to construct a labelled definite clause $x : H$ which when added to S gives the SLP S'. The clause H must be such that

$$B \wedge H \models E.$$

Suppose $x : H$ is placed within definition S'_q. By replacing all other labels y_i in S_q by $y_i(1-x)$ in S'_q we can ensure that π_q remains 1. The label x is chosen to maximise the likelihood $p(E|S')$ where

$$p(E|S') = \prod_{e \in E} \sum_{p \in SS(e,S')} \prod_{l \text{ in } p} l \tag{1}$$

and $SS(e, S')$ represents the set of SSLD derivations of e from S'.

4 Optimal parameter choice

Remark 4 *Assuming that S is non-recursive, the term $\prod_{l \in p} l$ either has the form*

c *(where c is a constant) if proof p does not involve a q clause,*

cx *if proof p involves $x : H$ or*

c(1-x) *if proof p involves a clause other than $x : H$ in q.*

If we vary the probability label x then $p(E|S')$ is maximal when

$$\begin{aligned}\frac{\mathrm{d}\ p(E|S')}{dx} &= 0 \text{ and} \qquad (2)\\ \frac{\mathrm{d}^2 p(E|S')}{dx^2} &\leq 0 \qquad (3)\end{aligned}$$

Unfortunately the term-size of a differential of a product of sums of products increases rapidly in the number of summed terms due to the form of the product rule. We take the alternative route of differentiating ln $p(E|S')$. The following theorem allows us to identify the solution of Equations (2) and (3) with the solution for the maximum of ln $p(E|S')$.

Remark 5 *Every differentiable function $f(x)$ is maximal when $\frac{d\ ln\ f(x)}{dx} = 0$ and $\frac{d^2 ln\ f(x)}{dx^2} \leq 0$.*

4.1 General case

Given Remark 5 we now log transform Equation (1) as follows.

$$\begin{aligned}\ln \mathrm{p}(E|S') &= \sum_{e\in E} \ln \left[\sum_{p\in SS(e,S')} \prod_{l\in p} l \right]\\ &= \sum_{e\in E} \ln U(e,x)\\ \frac{\mathrm{d}\ \ln \mathrm{p}(E|S')}{dx} &= \sum_{e\in E} \frac{\mathrm{d}\ \ln U(e,x)}{dx}\\ &= \sum_{e\in E} \frac{1}{U(e,x)} \frac{\mathrm{d}\ \mathrm{U}(e,x)}{dx} \qquad (4)\end{aligned}$$

From Remark 4 above we get the following.

$$U(e,x) = x(c_1 + c_2 + \ldots) + (1-x)(d1 + d2 + \ldots) + c(e) \qquad (5)$$

where c_i, d_j are products of probability labels from S and $c(e)$ is a sum of products of labels from S. We can now simplify Equation (5) as follows.

$$U(e,x) = xk_1(e) + k_2(e) \qquad (6)$$

where $c = c_1 + c_2 \ldots$, $d = d_1 + d_2 \ldots$, $k_1(e) = (c - d)$, $k_2(e) = d + c(e)$. Combining Equation (5) and (6) gives the following.

$$\begin{aligned}
\frac{\mathrm{d}\ \ln\ \mathrm{p}(E|S')}{dx} &= \sum_{e \in E} \frac{k_1(e)}{xk_1(e) + k_2(e)} \\
&= \sum_{e \in E} \frac{1}{x + \frac{k_2(e)}{k_1(e)}} \\
&= \sum_{e \in E} \frac{1}{x + k(e)} && (7) \\
\frac{d^2\ \ln\ p(E|S')}{dx^2} &= -\sum_{e \in E} \frac{1}{(x + k(e))^2} \leq 0 && (8)
\end{aligned}$$

where $k(e) = \frac{k_2(e)}{k_1(e)}$.

The following theorem defines the general case for finding the value of x which maximises $p(E|S')$.

Theorem 6 *If $p(E|S) \neq 0$ then $p(E|S')$ is maximal when x takes a value defined by $\sum_{e \in E} \frac{1}{x+k(e)} = 0$.*

Proof. *Follows trivially from Remark 5 and Equation (7)naly together with the fact that $\frac{d^2 \ln\ P(E|S')}{dx^2} \leq 0$ due to the form of Equation (8).*

4.2 Analytical solution for two example case

We now demonstrate how the general equation from Theorem 6 can be solved analytically for the case in which E contains only two examples, e_1 and e_2. In terms of ILP this corresponds to the case of finding the probability label for the least general generalisation of two examples. In this case

$$\begin{aligned}
\sum_{e \in E} \frac{1}{x + k(e)} &= 0 \\
\Rightarrow (x + k(e_1)) + (x + k(e_2)) &= 0 \\
\Rightarrow x &= -\left[\frac{k(e_1) + k(e_2)}{2}\right]. && (9)
\end{aligned}$$

We demonstrate this analytical solution below by way of an example.

Example 7 *Let S' be*

$$S' = \left\{ \begin{array}{lll} x: & p(X,Y) \leftarrow q(X,Z), r(Z,Y) & [A] \\ 1-x: & p(X,Y) \leftarrow r(X,Z), s(Y,Z) & [B] \\ & & \\ 0.3: & q(a,b) \leftarrow & [C] \\ 0.4: & q(b,b) \leftarrow & [D] \\ 0.3: & q(c,e) \leftarrow & [E] \\ & & \\ 0.4: & r(b,d) \leftarrow & [F] \\ 0.6: & r(e,d) \leftarrow & [G] \\ & & \\ 0.9: & s(d,d) \leftarrow & [H] \\ 0.1: & s(e,d) \leftarrow & [I] \end{array} \right.$$

and E be

$$E = \left\{ \begin{array}{ll} p(b,d) \leftarrow & [e_1] \\ p(c,d) \leftarrow & [e_2] \end{array} \right.$$

Now the proofs for e_1 and e_2 are as follows.

$$SSLD(e_1, S) = \{\langle A, D, F\rangle, \langle B, F, H\rangle\}$$
$$SSLD(e_2, S) = \{\langle A, E, G\rangle, \langle B, G, I\rangle\}$$

Given these proofs the likelihood function is as follows.

$$p(E|S) = [x(0.4)(0.4) + (1-x)(0.4)(0.9)] * [x(0.3)(0.6) + (1-x)(0.6)(0.1)]$$

This function is plotted in Figure 1. Now for e_1 we have the following.

$$\begin{aligned} U(e_1, x) &= x(c_1 + c_2 + \ldots) + (1-x)(d_1 + d_2 + \ldots) + c(e_1) \\ c &= c_1 = (0.4)(0.4) = 0.16 \\ d &= d_1 = (0.4)(0.9) = 0.36 \\ c(e_1) &= 0 \\ k_1(e_1) &= (c - d) = -0.20 \\ k_2(e_1) &= (d + c(e_1)) = 0.36 \\ k(e_1) &= \frac{k_2(e_1)}{k_1(e_1)} = \frac{0.36}{-0.20} = -1.8 \end{aligned}$$

By similar reasoning we have the following for e_2.

$$k(e_2) = \frac{k_2(e_2)}{k_1(e_2)} = \frac{0.06}{0.12} = 0.5$$

Thus according to Equation (9) the optimal value for x is as follows.

$$\begin{aligned} x &= -\frac{k(e_1) + k(e_2)}{2} \\ &= 0.65 \end{aligned}$$

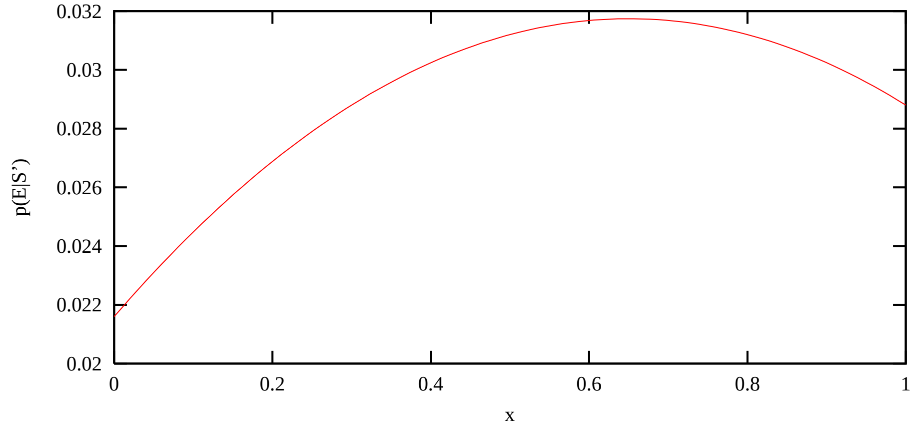

Figure 1: Likelihood function for Example 7

4.3 Finding descending intervals for n example case

Analytical solutions become complex to solve for large n. However, the following two properties of $\ln\, p(E|S')$ simplify the finding of numerical solutions.

Remark 8 *In Equation 7, there exists a singularity whenever $x = -k(e)$. The function ln $p(E|S')$ tends to ∞ and $-\infty$ as the singularity is approached from above and below respectively.*

Theorem 9 *Suppose $K(E) = \{k(e_1, \ldots, k(e_n))\}$ is an ordered set representing the $k(e)$ values from Equation 7. Equation 7 has $n-1$ roots of the form $-k(e_i) < r_i < -k(e_{i+1})$.*

Proof. *Equation 7 can be rewritten as a polynomial function of x of order n, and therefore has $n-1$ roots. The function ln $p(E|S')$ varies smoothly from ∞ at $-k(e_i)$ to $-\infty$ at $-k(e_{i+1})$. This follows from Remark 8 and the fact that Equation 8 shows that ln $p(E|S')$ is everywhere descending. From this it follows that for each i, $1 \leq n-1$, there must exist a value $x = r_i$ where $-k(e_i) < r_i < -k(e_{i+1})$ such that ln $p(E|S') = 0$. By definition these are the $n-1$ roots of Equation 8.*

This theorem provides the basis for finding the set of descending intervals containing the optimal probability label for the new clause.

Example 10 *Suppose $K(E) = \{k(e_1), \ldots, k(e_4)\} = \{-0.1, -0.3, -0.5, -0.8\}$. Figure shows the associated graph of $\frac{d\ \ln\, p(E|S')}{dx}$. The 3 roots are found at $0.1 < r_1 < 0.3 < r_2 < 0.5 < r_3 < 0.8$.*

Having found the intervals in which optimal solutions exist, an efficient numerical algorithm can be used to find the roots.

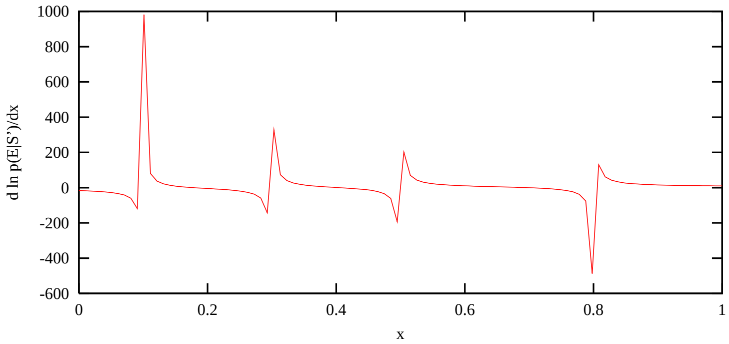

Figure 2: Values on the graph at which $\frac{\mathrm{d}\ \ln\ \mathrm{p}(E|S')}{dx} = 0$ indicated the optimal values of the probability label given $K(E)$ in Example 10.

4.4 Newton's method for finding roots

Newton's method for finding roots is based on using the first derivative of a function $g(x)$ to derive a convergent series of guesses r_0, r_1 for the roots of $g(x)$. The convergent series is derived using the following recurrence equation.

$$r_{i+1} = r_i - \frac{g(r_i)}{g'(r_i)} \tag{10}$$

A critical choice necessary for using Newton's method is the choice of r_0. Assuming multiple roots, the root found will be that closest to r_0. Given Theorem 9 an obvious choice for r_0 is the series of mid-points of the $n-1$ intervals defined by $K(E)$.

Example 11 *As with Example 10 we suppose* $K(E) = \{k(e_1), \ldots, k(e_4)\} = \{-0.1, -0.3, -0.5, -0.8\}$. *Applying Equations (eqn6) and (eqn6.1) give the following.*

$$g(x) = \sum_{e \in E} \frac{1}{x + k(e)} \tag{11}$$

$$g'(x) = -\sum_{e \in E} \frac{1}{(x + k(e))^2} \tag{12}$$

Using Equation (10) gives the following convergent sequence for the root found in the first interval.

$$\begin{aligned} r_0 &= 0.2 \\ r_1 &= 0.1766233766233766 \\ r_2 &= 0.1776269717216822 \end{aligned}$$

$$
\begin{aligned}
r_3 &= 0.1776336478800729 \\
r_4 &= 0.1776336481635912 \\
r_5 &= 0.1776336481635912
\end{aligned}
$$

In the following sections we describe a method related to that described in [] in which optimal probability label computation has been incorporated into Progol.

5 Overview of Progol

ILP systems take LPs representing background knowledge B and examples E and attempt to find the simplest consistent hypothesis H such that the following holds.

$$B \wedge H \models E \tag{13}$$

This section briefly describes the Mode Directed Inverse Entailment (MDIE) approach used in Progol [6]. Equation 13 is equivalent for all B, H and E to the following.

$$B \wedge \overline{E} \models \overline{H}$$

Assuming that $\overline{H}$ and $\overline{E}$ are ground and that $\overline{\bot}$ is the conjunction of ground literals which are true in all models of $B \wedge \overline{E}$ we have the following.

$$B \wedge \overline{E} \models \overline{\bot}$$

Since $\overline{H}$ is true in every model of $B \wedge \overline{E}$ it must contain a subset of the ground literals in $\overline{\bot}$. Hence

$$B \wedge \overline{E} \models \overline{\bot} \models \overline{H}$$

and so for all H

$$H \models \bot \tag{14}$$

The set of solutions for H considered by Progol is restricted in a number of ways. Firstly, $\bot$ is assumed to contain only one positive literal and a finite number of negative literals. The set of negative literals in $\bot$ is determined by mode declarations (statements concerning the input/output nature of predicate arguments and their types) and user-defined restrictions on the depths of variable chains.

Progol uses a covering algorithm which repeatedly chooses an example e, forms an associated clause $\bot$ and searches for the clause which maximises the information compression within the following bounded sub-lattice.

$$\Box \preceq H \preceq \bot$$

The hypothesised clause H is then added to the clause base and the examples covered by H are removed. The algorithm terminates when all examples have been covered. In the original version of Progol (CProgol4.1) [6] the search for each clause H involves maximising the 'compression' function

$$f = (p - (c + n + h))$$

where p and n are the number of positive and negative examples covered by H, c is the number of literals in H, and h is the minimum number of additional literals required to complete the input/output variable chains in H (computed by considering variable chains in $\perp$). In later versions of Progol the following function was used instead to reduce the degree of greediness in the search.

$$f = \frac{m}{p}(p - (c + n + h)) \tag{15}$$

This function estimates the overall global compression expected of the final hypothesised set of clauses, extrapolated from local coverage and size properties of the clause under construction. A hypothesised clause H is pruned, together with all its more specific refinements, if either

$$1 - \frac{c}{p} \leq 0 \tag{16}$$

or there exists a previously evaluated clause H' such that H' is an acceptable solution (covers below the noise threshold of negative examples and the input/output variable chains are complete) and

$$1 - \frac{c}{p} \leq 1 - \frac{c' + n' + h'}{p'} \tag{17}$$

where p, c are associated with H and p', n', c', h' are associated with H'.

6 User-defined evaluation in Progol4.5

User-defined evaluation functions in Progol4.5 are implemented by allowing redefinition in Prolog of p, n and c from Equation 15. Figure 3 shows the convention for names used in Progol4.5 for the built-in and user-defined functions for these variables. Though this approach to allowing definition of the evaluation function is indirect, it means that the general criteria used in Progol for pruning the search (see Inequalities 16 and 17) can be applied unaltered as long as user_pos_cover and user_neg_cover monotonically decrease and user_hyp_size monotonically increases with downward refinement (addition of body literals) to the hypothesised clause. For learning SLPs these functions are derived below.

6.1 Bayes' function

We now reconsider the generalisation model described in Section 3 in the context of a search in which the hypothesised clause H varies. In each case the clause

Variable	Built-in	User-defined
p	pos_cover(P1)	user_pos_cover(P2)
n	neg_cover(N1)	user_neg_cover(N2)
c	hyp_size(C1)	user_hyp_size(C2)
h	hyp_rem(H1)	user_hyp_rem(H2)

Figure 3: Built-in and user defined predicates for some of the variables from Equation 15.

$x : H$ which is added to S to give S' has an optimal value of x chosen acording to the approach described in Sections 4.3 and 4.4. The posterior probability of S' given E can be expressed using Bayes' theorem as follows.

$$p(S'|E) = \frac{p(S')p(E|S')}{p(E)} \tag{18}$$

$p(S')$ represents a prior probability distribution over SLPs. If we suppose (as is normal) that the e_i are chosen randomly and independently from some distribution D over the instance space X then $p(E|S') = \prod_{i=1}^{m} p(e_i|S')$. We assume that $p(e_i|S') = Q(e_i|S')$ (see Section 2.2). $p(E)$ is a normalising constant. Since the probabilities involved in the Bayes' function tend to be small it makes sense to re-express Equation 18 in information-theoretic terms by applying a negative log transformation as follows.

$$-log_2 p(S'|E) = -log_2 p(S') - \sum_{i=1}^{m}[log_2 p(e_i|S')] + c \tag{19}$$

Here $-log_2 p(S')$ can be viewed as expressing the size (number of bits) of S'. The quantity $-\sum_{i=1}^{m}[log_2 p(e_i|S')]$ can be viewed as the sum of sizes (number of bits) of the derivations of each e_i from S'. c is a constant representing $log_2 p(E)$. Note that this approach is similar to that described in [9], differing only in the definition of $p(e_i|S')$. The approach in [9] uses $p(e_i|S')$ to favour LP hypotheses with low generality, while Equation 19 favours SLP hypotheses with a low mean derivation size. Surprisingly this makes the Bayes' function for learning SLPs appropriate for finding LPs which have low time-complexity with respect to the examples. For instance, this function would prefer an SLP whose underlying LP represented quick-sort over one whose underlying LP represented insertion-sort since the mean proof lengths of the former would be lower than those of the latter.

Equation 19 can be rewritten in terms of an information function I as

$$I(S'|E) = I(S') - \sum_{i=1}^{m} I(e_i|S') + c \tag{20}$$

where $I(x) = -\log_2 x$. The degree of compression achieved by an hypothesis is computed by subtracting $I(S'|E)$ from $I(S' = E|E)$, the posterior information of the hypothesis consisting of returning ungeneralised examples.

$$\begin{aligned} I(S' = E|E) &= I(E) + I(E|S' = E) + c \\ &= m + m\log_2 m + c \\ &= m(1 + \log_2 m) + c \end{aligned} \tag{21}$$

The compression induced by S' with respect to E is now simply the difference between Equations 21 and 20, which is as follows.

$$\begin{aligned} & m(1 + \log_2 m) - I(S') + \sum_{i=1}^{m} I(e_i|S') \\ = \ & \frac{m}{p}(p(1 + \log_2 m) - I(H) + \sum_{j=1}^{p} I(e_j|H)) \end{aligned} \tag{22}$$

In Equation 22 extrapolation is made from the p positive examples covered by hypothesised clause H. Comparing Equations 15 and 22 the user-defined functions of Figure 3 are as follows (p, n, c, h represent built-in functions and p', n', c', h' represent their user-defined counter-parts).

$$\begin{aligned} p' &= p(1 + \log_2 m) \\ n' &= \sum_{j=1}^{m} I(e_j|H) + n \\ c' &= c \\ h' &= h \end{aligned} \tag{23}$$

7 Discussion of related work

This section describes some of the related approaches which have been taken to learning probabilistic logic representations.

7.1 Learning PRMS

PRMs share the underlying probabilistic semantics and local independence assumptions of Bayesian Networks. This has allowed many of the Bayesian net learning techniques to be extended to PRMs. For instance, Koller and Pfeffer [5] have used EM (Expectation Maximisation [3]) to estimate the parameters θ_S of a PRM for which the dependency structure is known. Unlike the algorithms described in the present paper, EM is not guaranteed to converge to optimal values. However, the multi-variate parameter estimation problem being attacked by Koller and Pfeffer using EM is considerably harder than the univariate problem considered here. More recently Friedman et al. [4] have also attacked the more difficult problem of learning the dependency structure S directly from data.

7.2 Learning SLPs

The task of learning SLPs, like that of learning PRMs, has previously been divided into that of parameter estimation and structure learning. Cussens [2] presents an algorithm called Failure-Adjusted Maximisation (FAM) which estimates from data the parameters of an SLP for which the underlying logic program is given. FAM is an instance of the EM algorithm that applies specifically to normalised SLPs. Recently the author [12] presented a two-phase algorithm for learning both the parameters of an SLP and the underlying logic program from data. The algorithm is based on maximising Bayes' posterior probability, and has been demonstrated on problems involving learning an animal taxonomy and a simple English grammar.

8 Conclusions and further work

In this paper we have considered ways in which both the structure and the parameters of an SLP can be learned simultaneously. The paper assumes an ILP algorithm, such as Progol or FOIL, in which clauses are constructed independently. We have derive analytical and numerical methods for efficient computation of the optimal probability parameters for a single clause choice within such a search.

Further analysis is required for some of the approaches described in this paper. For instance, the simplifying assumption made in Remark 4 that S be non-recursive may be overly-restrictive in some cases, though it is almost identical to the C-derivation assumption used in relative least generalisation [13, 10]. Also the convergence rate of the iteration method described in Section 4.3 needs to be analysed, and compared to alternatives approaches. In particular, empirical comparisons should be made against Cussen's FAM algorithm [2] when the latter is restricted to the case of single label estimation.

Further work is required to test the approach described in this paper. The performance of this new version of Progol needs to be compared with the two stage implementation described in [8]. The author believes that the new approach has the potential to improve on that described in [8] in terms of both efficiency and accuracy of the generated result.

Acknowledgements

Many thanks are due to my wife, Thirza and daughter Clare for the support and happiness they give me. This work was supported partly by the ESPRIT IST project "Application of Probabilistic Inductive Logic Programming (APRIL)", the EPSRC grant "Closed Loop Machine Learning" and the BBSRC/EPSRC Bio-informatics and E-Science Programme, "Studying Biochemical networks using probabilistic knowledge discovery".

References

[1] J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 126–133, San Francisco, 1999. Kaufmann.

[2] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[3] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[4] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI-99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1309, San Mateo, CA:, 1999. Morgan-Kaufmann.

[5] D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1316–1321, San Mateo, CA:, 1997. Morgan-Kaufmann.

[6] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[7] S. Muggleton. Inductive logic programming: issues, results and the LLL challenge. *Artificial Intelligence*, 114(1–2):283–296, December 1999.

[8] S. Muggleton. Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, 5(041), 2000.

[9] S. Muggleton. Learning from positive data. *Machine Learning*, 2001. Accepted subject to revision.

[10] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381, Tokyo, 1990. Ohmsha.

[11] S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

[12] S.H. Muggleton. Learning stochastic logic programs. In Lise Getoor and David Jensen, editors, *Proceedings of the AAAI2000 workshop on Learning Statistical Models from Relational Data*. AAAI, 2000.

[13] G. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press, 1971.

[14] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.