Statistical Aspects of Stochastic Logic Programs

James Cussens
Department of Computer Science, University of York
Heslington, York, YO10 5DD, UK
jc@cs.york.ac.uk

Abstract

Stochastic logic programs (SLPs) and the various distributions they define are presented with a stress on their characterisation in terms of Markov chains. Sampling, parameter estimation and structure learning for SLPs are discussed. The application of SLPs to Bayesian learning, computational linguistics and computational biology are considered. Lafferty's Gibbs-Markov models are compared and contrasted with SLPs.

1 INTRODUCTION

> ... I find nothing in logistic for the discoverer but shackles ... if it requires 27 equations to establish that 1 is a number, how many will it require to demonstrate a real theorem? (Poincaré, quoted in (Machover and Bell, 1977))

There is currently considerable interest amongst the AI community in developing probabilistic knowledge representations that extend existing approaches to incorporate domain knowledge and/or relational data (Ngo and Haddaway, 1997; Friedman et al., 1999; Kersting and De Raedt, 2000; Muggleton, 2000b; Cussens, 2000).

One approach to this problem is to integrate probabilistic and 'logical' methods: an idea that has a long history dating back to (Boole, 1854). The idea is to use a set of logical formulae to encode domain knowledge. Boole used propositional logic, but now we can use first-order logic to encode domain knowledge in a *first-order theory.* Such a theory can describe *relations* between objects. Probability can then be 'added' so that the probability with which an object has some attribute depends on the attributes which related objects have (Friedman et al., 1999) or, more generally, so that probabilities depend on what is entailed by the logically-encoded domain knowledge (Ngo and Haddaway, 1997).

Note that there is a long history in statistics of work on 'relational learning', i.e. probabilistic models where data are not independent and identically distributed. Consider a multivariate time series of some sort (e.g. ARIMA) for the analysis of, say, financial data. Here the objects are days and the 'attributes' of these objects might be the prices of various commodities on those days. In such a model, we have that, e.g. the price of pork-belly futures on day t is related (or more precisely correlated) with the exchange rate of the dollar on day $t-1$. In time series the relation between objects is one of temporal succession; in spatial statistics, we have more complex spatial relationships.

It is important that future work in the AI community on 'relational learning' takes advantage of this existing work on time series and spatial statistics, although this is not achieved here. My motivation for using logic to encode relations between objects is simply the flexibility it gives for expressing all sorts of relations, it is *not* motivated by a desire to formalise statistical inference. Although logical analysis is useful in clarifying foundations, there can be, as Poincaré noted, severe problems with its application to real problems.

This paper concerns stochastic logic programs (SLPs); one approach to effecting a marriage between logic and probability which follows this pragmatic approach to logic. The paper is organised as follows. Section 2 examines statistical aspects of SLPs, giving the various distributions that can be defined with an SLP and addressing sampling and parameter estimation, and structure learning. Section 3 argues that the distributions defined by SLPs are useful for a number of applications. Section 4 compares SLPs to Lafferty's Gibbs-Markov models and the paper then concludes with Section 5.

2 STATISTICAL ASPECTS OF SLPS

This section begins with definitions for various types of SLPs and describes the distributions defined by an SLP, paying particular attention to the connection with Markov chains.

Definition 1 *A* stochastic logic program (SLP) $\mathcal{S}$ *is a definite logic program where some of the clauses are parameterised with non-negative numbers. A* pure SLP *is an SLP where all clauses have parameters, as opposed to an* impure *SLP where not all clauses have parameters. A* normalised SLP *is one where parameters for clauses which share the same predicate symbol sum to one. If this is not the case, then we have an* unnormalised SLP*.*

In this paper we will restrict attention to pure normalised SLPs, since these have a nice characterisation in terms of Markov chains. Fig 1 shows $\mathcal{S}_0$, a very simple example of a pure normalised SLP. Note that the first clause in $\mathcal{S}_0$, although syntactically legal, would not be found in a real logic program, since it is logically equivalent to the simpler clause $s(X) \leftarrow p(X)$. However, the distributions defined by SLPs depend on the syntactic structure of the underlying logic program in such a way that replacing $0.4 : s(X) \leftarrow p(X), p(X)$ by $0.4 : s(X) \leftarrow p(X)$ would change the probability distributions defined by $\mathcal{S}_0$.

```
0.4:s(X) :- p(X), p(X).  0.3:p(a).  0.2:q(a).
0.6:s(X) :- q(X).        0.7:p(b).  0.8:q(b).
```

Figure 1: $\mathcal{S}_0$: A simple pure, normalised SLP

2.1 DISTRIBUTIONS DEFINED BY SLPS

SLPs associate probability distributions with *goals*. Unfortunately lack of space disallows a proper account of the role goals play in logic programming, so an informal description is given. The essentials are best understood by considering SLD-trees, an example of which is given in Fig 2. An SLD-tree is a search tree for refutations of the top-level goal at the root of the tree. In the case of Fig 2 the top level goal is `:- s(X)` which is Prolog notation for the first-order formula $\forall X \neg s(X)$. Essentially, a child of a goal is produced by unifying the leftmost atomic formula in the goal with the head of a clause in the SLP and replacing that leftmost atomic formula with the clause body. If the leftmost atomic formula fails to unify with a clause head despite them both sharing the same predicate symbol then a `fail` child is produced. Since there are two clause heads in $\mathcal{S}_0$ which unify with `s(X)`, the goal `:- s(X)` has two (non-failure) children. Any variable substitutions required to effect this unification are applied to the new goal. So, for example, unifying the leftmost `p(X)` of `:- p(X),p(X)` with clause head `p(a)` leaves the goal `:- p(a)`.

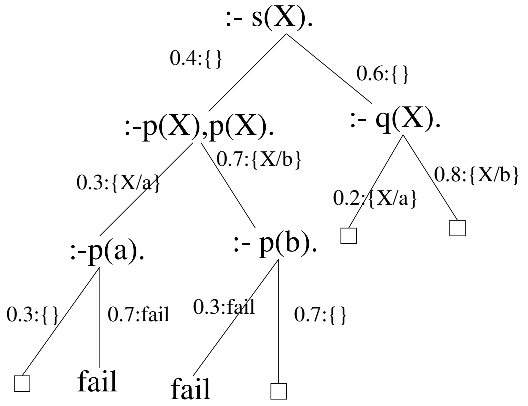

Figure 2: Annotated SLD-tree for $\mathcal{S}_0$

An *derivation* is a branch of the SLD-tree. A *refutation* is a derivation ending with the empty goal □; Fig 2 shows that there are four refutations of `:- s(X)`, two of which instantiate X to a and two which instantiate X to b. Note that Fig 2 also contains two failure derivations, the leftmost one of which corresponds to an attempt to unify `p(a)` with `p(b)`.

The execution of a normal logic program corresponds to a search for refutations in the SLD-tree; in standard Prolog this search is a depth-first, leftmost-first search. An SLP essentially replaces this deterministic exploration of the SLD-tree with a probabilistic one which we can describe in terms of Markov chains where the states of the chain are goals.

Definition 2 *Let $\mathcal{S}$ be a pure normalised SLP and let G_0 be a goal. Let l_i be the parameter associated with clause C_i. $\mathcal{S}$ and G_0 define a Markov chain where the states of the Markov chain are goals G_κ. The initial state probabilities are:*

$$a_\kappa = \begin{cases} 1 & \textit{if } G_\kappa = G_0 \\ 0 & \textit{otherwise} \end{cases}$$

and the transition probabilities are:

$$p_{\kappa\kappa'} = \begin{cases} l_i & \textit{if } G_{\kappa'} \textit{ is a child of} \\ & \quad G_\kappa \textit{ produced by using } C_i \\ 1 & \textit{if } G_\kappa = G_{\kappa'} = \Box \\ 1 & \textit{if } G_\kappa = G_{\kappa'} = \texttt{fail} \\ 0 & \textit{otherwise} \end{cases}$$

The absorbing states of the Markov chain are $\Box$ and `fail` and we identify finite derivations with infinite sequences in the Markov chain which reach either $\Box$ and `fail` and remain there. This identification allows the Markov chain to define a distribution over all derivations in the SLD-tree. There are three sorts of derivations: (i) finite derivations ending in $\Box$; (ii) finite derivations ending in `fail` and (iii) infinite derivations. This distribution over derivations is denoted $\psi_{(\lambda,\mathcal{S},G)}$ where G is the initial goal and λ is a vector composed of the logs of the parameters l_i. If $\nu_i(x)$ is the frequency with which clause C_i is used in a derivation x and $\nu(x)$ is the vector of all these n clause counts we have

$$\psi_{(\lambda,\mathcal{S},G)}(x) = \prod_{i=1}^{n} l_i^{\nu_i(x)} = e^{\lambda \cdot \nu(x)} \tag{1}$$

Let $R(G)$ be the set of all refutations of a goal G. Consider now the conditional distribution $f_{(\lambda,\mathcal{S},G)}$ defined as follows:

$$f_{(\lambda,\mathcal{S},G)}(x) \stackrel{\text{def}}{=} \psi_{(\lambda,\mathcal{S},G)}(x | x \in R(G))$$

Next define:

$$Z_{(\lambda,\mathcal{S},G)} \stackrel{\text{def}}{=} \sum_{x \in R(G)} \psi_{(\lambda,\mathcal{S},G)}(x) = \psi_{(\lambda,\mathcal{S},G)}(R(G))$$

$Z_{(\lambda,\mathcal{S},G)}$ is just the probability (according to $\psi_{(\lambda,\mathcal{S},G)}$) that a derivation of G is a refutation. For example we have $Z_{(\lambda,\mathcal{S}_0,\leftarrow s(X))} = 0.832$. Note that $Z_{(\lambda,\mathcal{S},\Box)} = 1$ and $Z_{(\lambda,\mathcal{S},\texttt{fail})} = 0$. For other goals we have

$$Z_{(\lambda,\mathcal{S},G)} = \sum l_i Z_{(\lambda,\mathcal{S},G')} \tag{2}$$

where the sum is over all G' which are children of G and where l_i is the parameter associated with C_i, the clause used to generate G' from G. It is useful to think of $Z_{(\lambda,\mathcal{S},G)}$ as the 'weight' of the subtree below G, where failure derivations have zero weight.

We now have that

$$f_{(\lambda,\mathcal{S},G)}(x) = \begin{cases} Z^{-1}_{(\lambda,\mathcal{S},G)} \psi_{(\lambda,\mathcal{S},G)}(x) & \text{if } x \in R(G) \\ 0 & \text{if } x \notin R(G) \end{cases}$$

so for refutations r we have from (1)

$$f_{(\lambda,\mathcal{S},G)}(r) = Z^{-1}_{(\lambda,\mathcal{S},G)} e^{\lambda \cdot \nu(r)}$$

$f_{(\lambda,\mathcal{S},G)}$ is then a loglinear model with the clause frequencies $\nu(r)$ as features. We can also define $f_{(\lambda,\mathcal{S},G)}$ in terms of a Markov chain. As before we have:

$$a_\kappa = \begin{cases} 1 & \text{if } G_\kappa = G_0 \\ 0 & \text{otherwise} \end{cases}$$

but now the transition probabilities are:

$$p_{\kappa\kappa'} = \begin{cases} \frac{Z_{(\lambda,\mathcal{S},G_{\kappa'})}}{Z_{(\lambda,\mathcal{S},G_\kappa)}} l_i & \text{if } G_{\kappa'} \text{ is a child of} \\ & \quad G_\kappa \text{ produced by using } C_i \\ 1 & \text{if } G_\kappa = G_{\kappa'} = \Box \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Having now defined $\psi_{(\lambda,\mathcal{S},G)}$ and $f_{(\lambda,\mathcal{S},G)}$ it is possible to define $p_{(\lambda,\mathcal{S},G)}$ the third sort of distribution defined by an SLP. First recall that each refutation instantiates the variables in the initial top-level goal. These instantiations are known as computed answers and we can use them to define the *yields* of refutations as follows:

Definition 3 *The* yield $Y(r)$ *of a refutation* r *of a unit goal* $G = \leftarrow A$ *is* $A\theta$ *where* θ *is the computed answer for* G *using* r*. The set of* proofs *for an atom* y_k *is the set* $X(y_k) = \{r | Y(r) = y_k\}$.

An SLP defines a distribution over these yielded atoms by simple marginalisation:

$$p_{(\lambda,\mathcal{S},G)}(y_k) \stackrel{\text{def}}{=} \sum_{r \in X(y_k)} f_{(\lambda,\mathcal{S},G)}(r)$$

In the case of $\mathcal{S}_0$, there are two refutations of `:- s(X)` that yield the atom $s(a)$ and two that yield $s(b)$. We have

$$\begin{aligned} p_{(\lambda,\mathcal{S}_0,\leftarrow s(X))}(s(a)) &= (0.036 + 0.12)/0.832 = 0.1875 \\ p_{(\lambda,\mathcal{S}_0,\leftarrow s(X))}(s(b)) &= (0.196 + 0.48)/0.832 = 0.8125 \end{aligned}$$

The distributions $\psi_{(\lambda,\mathcal{S},G)}$, $f_{(\lambda,\mathcal{S},G)}$ and $p_{(\lambda,\mathcal{S},G)}$ are all essentially defined in terms of making probabilistic choices when moving *down* the SLD-tree. Because of this they can be defined in terms of goals rather than nodes in the SLD-tree. If the logic program underlying the SLP is recursive, it may be that the same goal occurs more than once in the SLD-tree. However, the SLD-subtree underneath a goal is identical wherever a goal appears in the tree, and if we are always travelling down the tree we can ignore at which particular node in the tree the goal appears.

In (Cussens, 2000), in contrast, a Markov chain is defined using an SLP which jumps between leaves of the SLD-tree by first backtracking n steps with (roughly) probability $p^n(1-p)$ and thus arriving at an interior node of the tree and then probabilistically moving down the tree (according to $\psi_{(\lambda,\mathcal{S},G)}$) until a leaf is reached. In this case the states of the Markov chain are nodes of the SLD-tree rather than goals. Fig 3 illustrates a proposed transition from leaf M^i to leaf

M^* via node G where we backtrack through two choice points ($n^i = 2$) and then move down through two choice points ($n^* = 2$). The parameter of the first clause used to reach M^i from G is l^i, and l^* is defined similarly.

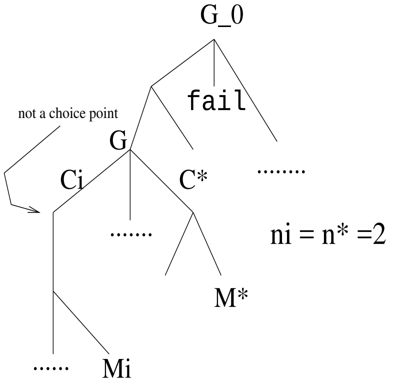

Figure 3: Jumping from M^i to M^* in the SLD-tree

The purpose of the Markov chain is to explore a posterior distribution over some space of models (i.e. MCMC), where the prior distribution has been defined by $p_{(\lambda,\mathcal{S},G)}$. Each leaf node corresponding to a refutation is assumed to yield an atom representing some model in the model space. Failure derivations are identified with zero likelihood models. Let $\ell(M^*)$ denote the likelihood of the model at leaf M^*, then a proposed transition from leaf M^i to leaf M^* is accepted with probability $\alpha(M^i, M^*)$ where:

$$\alpha(M^i, M^*) = \min\left\{ p^{(n^*-n^i)} \frac{1-l^i}{1-l^*} \frac{\ell(M^*)}{\ell(M^i)}, 1 \right\}$$

This defines yet another Markov chain but one where $\Box$ and `fail` are not absorbing.

2.2 SAMPLING FROM SLPS

Sampling from $p_{(\lambda,\mathcal{S},G)}$ amounts to running the Markov chain associated with $f_{(\lambda,\mathcal{S},G)}$ and then just remembering the substitutions that were made along the way. As Section 3 argues, $p_{(\lambda,\mathcal{S},G)}$ is the most useful distribution and as shown later in this section, efficient sampling from $p_{(\lambda,\mathcal{S},G)}$ is crucial for parameter estimation for SLPs.

Unfortunately, sampling from $p_{(\lambda,\mathcal{S},G)}$ is hard because the $\frac{Z_{(\lambda,\mathcal{S},G_{\kappa'})}}{Z_{(\lambda,\mathcal{S},G_{\kappa})}}$ values can not usually be easily calculated from the clause parameters l_i. One option is to approximate this ratio and use an importance sampling approach to compensate for the approximation. Such an approach is described in (Cussens, 2000) which also describes a method for exact computation of Z values using (2).

2.3 PARAMETER ESTIMATION IN SLPS

Suppose we have data in the form of a sequence of atomic formulae which we take to have been generated by $p_{(\lambda,\mathcal{S},G)}$ where the parameters λ are unknown. Assume that $\mathcal{S}$ and G are fixed so that $\psi_{(\lambda,\mathcal{S},G)}$ is abbreviated to ψ_λ. It is possible to apply the EM algorithm to do maximum likelihood estimation for λ by taking the dataset of atomic formulae to be the result of *truncating* and then *grouping* a hidden dataset of derivations generated according to ψ_λ. This is done by positing the following sampling mechanism. Derivations are sampled according to ψ_λ, these are then truncated by throwing away all the failure derivations leaving only refutations. The refutations are then grouped together according to their yields so that only the yielded atoms are observed.

(Dempster et al., 1977) show how to apply the EM algorithm to grouped and truncated data. Details of its application to SLPs (called *failure-adjusted maximisation (FAM)*) are given in (Cussens, 2001). Here we just outline how to compute $\psi_{\lambda^{(h)}}[\nu_i|y]$ the expected frequency according to the current parameters $\lambda^{(h)}$ with which clause C_i 'fired' while producing the data y. This is just a weighted sum of $\psi_{\lambda^{(h)}}[\nu_i|y_k]$ the clause's expected frequency when producing each of the observed atoms y_k plus $\psi_{\lambda^{(h)}}[\nu_i|\texttt{fail}]$ its expected frequency when generating failure derivations. This sum is given in (4) where N_k is the frequency of y_k in the data, and N is the size of the data.

$$\begin{aligned} \psi_{\lambda^{(h)}}[\nu_i|y] &= \sum_k N_k \psi_{\lambda^{(h)}}[\nu_i|y_k] + \\ & \quad N(Z^{-1}_{\lambda^{(h)}} - 1)\psi_{\lambda^{(h)}}[\nu_i|\texttt{fail}] \end{aligned} \tag{4}$$

The practicality of applying EM to SLPs depends on the computation or accurate estimation of the expectations in (4). If an SLP is failure-free then the second term in (4) disappears and the SLP is essentially an SCFG. The inside-outside algorithm can then be applied, or possibly even the forward-backward algorithm if the SLP represents a HMM. Note that $(Z^{-1}_{\lambda^{(h)}} - 1)$ *quantifies* the degree to which an SLP diverges from an SCFG—it is a measure of 'non-context-freeness'. In general, context-free methods will not suffice and it may be possible to adapt the tabular approach of (Kameya and Sato, 2000) which is applied to parameter estimation for PRISM models. Another, appealingly simple, approach is to estimate expectations by sampling from $\psi_{\lambda^{(h)}}$. An estimate of $Z_{\lambda^{(h)}}$ can be obtained by counting how often derivations turn out to be refutations.

2.4 LEARNING THE STRUCTURE OF SLPS

Inductive logic programming (ILP) is the area of machine learning concerned with the induction of logic programs from background knowledge and data. Since the structure of an SLP is simply a logic program, it follows that ILP techniques can be used in SLP structure learning. Indeed, (Muggleton, 2000a) has already done this, using the ILP algorithm Progol to learn the structure of an SLP and then obtaining a rough but quickly calculable estimate for the parameters.

Recall that $f_{(\lambda,\mathcal{S},G)}$ is a loglinear distribution so work on learning the structure (or *features*) of loglinear models can be applied to SLPs. In (Della Pietra et al., 1997) greedy feature selection is intertwined with parameter estimation. In SLPs, features are essentially clauses, so an adaptation of the algorithm of (Della Pietra et al., 1997) can use ILP to construct clauses. (Dehaspe, 1997) is related work in this direction.

3 APPLICATIONS OF SLPS

SLPs are useful when the flexibility of defining a distribution over first-order terms using $p_{(\lambda,\mathcal{S},G)}$ can be exploited. This is the case when defining priors over the structure of statistical models. For example, one can define a space of Bayesian nets using a logic program and then parameterise that logic program to give an SLP which defines a prior distribution over that space. MCMC can then be used, as sketched in Section 2.1 and described in (Cussens, 2000) to explore the posterior distribution.

The flexibility of first-order representations has long been exploited in computational linguistics. (Muggleton, 1996) explicitly introduced SLPs as generalisations of Hidden Markov models (HMMs) and Stochastic Context-Free Grammars (SCFGs). Statistical approaches, often using HMMs and SCFGs, have revolutionised computational linguistics (Hirschberg, 1998). More recently, there has been work using Maximum Entropy methods (i.e. loglinear models) applied to non-context-free models (Abney, 1997; Riezler, 1998). SLPs fall into this non-context-free category, and can, for example, been seen as a special case of Riezler's Probabilistic Constraint Logic Programs (Riezler, 1998) which Riezler uses to represent constraint grammars. Given that there is strong evidence that *natural language is not context-free*, it is important that research effort is focussed on these more complex statistical linguistic models to advance further the statistical NLP revolution.

Another potential application area is biological sequence analysis given the successful application of HMMs and SCFGs there. However, (Durbin et al., 1998) sound a note of caution concerning more complex models such as SLPs:

> We will not explore stochastic context-sensitive or stochastic unrestricted grammars in any detail, as we are unaware of any practical applications of these in computational biology

4 SLPS AND GIBBS-MARKOV MODELS

(Lafferty, 1996) notes that

> Standard statistical approaches to speech and language processing problems use hidden Markov models, or ... stochastic context-free grammars ... But such models are limited in their ability to incorporate contextual information and long-distance dependencies. Because of the Markov assumption, all predictive information must be encoded in the states.

(Lafferty, 1996) proposes Gibbs-Markov models (GMMs) to overcome this limitation. These models have an underlying HMM or PCFG, but state transition and output symbol probabilities are given by Gibbs distributions. If $\mathcal{H}_t$ is the vector of states visited and symbols output prior to time t then $P(S_t|\mathcal{H}_t)$ the probability of arriving at state S_t is given by:

$$P(S_t|\mathcal{H}_t) = \frac{1}{Z(\mathcal{H}_t)} e^{\lambda \cdot f(S_t,\mathcal{H}_t)}$$

where $f(S_t,\mathcal{H}_t)$ is a vector of binary feature values where each value is zero or one. (Lafferty, 1996) shows how the EM algorithm and generalized iterative scaling can be used for parameter estimation for GMMs and also describes a GMM for statistical language modelling.

Both GMMs and SLPs have an embedded context-free grammar. In the case of SLPs this is the failure-free SLP (equivalent to an SCFG) which can be produced by replacing each variable by a new distinct variable, so that $s(X) \leftarrow p(X), p(X)$ becomes $s(X1) \leftarrow p(X2), p(X3)$.

In contrast to GMMs, state transitions associated with $f_{(\lambda,\mathcal{S},G)}$ depend not on the past history of states but on the set of possible future states. For a goal G_κ these are the goals in the subtree with G_κ at its root—recall the definition of $p_{\kappa\kappa'}$ given in (3). Also in contrast to GMMs, context-dependence is achieved due to the possibility of failure. Indeed, as shown in the presentation of the FAM algorithm, $Z^{-1}_{(\lambda,\mathcal{S},G)}$ quantifies the degree of context-dependence.

5 CONCLUSIONS

In this paper a number of important issues have been omitted such as the incorporation of logically encoded domain knowledge using impure SLPs and the connections with (i) Bayesian nets and (ii) alternative logical-statistical frameworks. Despite this, there has been provided the basic statistical properties of SLPs. From a statistical point of view perhaps the most useful property of SLPs is their ability to define Markov models with a complex space of states defined via a very well understood formalism: first-order logic. The connection with logic *programming* will hopefully provide the computational and implementational techniques required for SLPs to become usable (and used) statistical models.

Acknowledgements

Thanks to 3 anonymous referees for useful comments.

References

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics*, 23(4):597–618.

Boole, G. (1854). *An Investigation of the Laws of Thought, on which are founded the Mathematical Theories of Logic and Probabilities.* Dover.

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pages 115–122, San Francisco, CA. Morgan Kaufmann Publishers.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning.* To appear.

Dehaspe, L. (1997). Maximum entropy modeling with clausal constraints. In *Inductive Logic Programming: Proceedings of the 7th International Workshop (ILP-97). LNAI 1297*, pages 109–124. Springer.

Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning probabilistic relational models. In *Proc. IJCAI-99.*

Hirschberg, J. (1998). Every time I fire a linguist, my performance goes up, and other myths of the statistical natural language processing revolution. In *Proc. AAAI-98.* Invited Talk.

Kameya, Y. and Sato, T. (2000). Efficient EM learning with tabulation for parameterized logic programs. In *Proceedings of the First International Conference on Computational Logic (CL 2000)*, volume 1861 of *LNAI*, pages 269–284, London. Springer.

Kersting, K. and De Raedt, L. (2000). Bayesian logic programs. In Cussens, J. and Frisch, A., editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155.

Lafferty, J. D. (1996). Gibbs-Markov models. *Computer Science and Statistics*, 27:370–377.

Machover, M. and Bell, J. (1977). *A Course in Mathematical Logic.* North-Holland, Amsterdam.

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

Muggleton, S. (2000a). Learning stochastic logic programs. In Getoor, L. and Jensen, D., editors, *Proceedings of the AAAI2000 workshop on Learning Statistical Models from Relational Data.*

Muggleton, S. (2000b). Semantics and derivation for stochastic logic programs. In *UAI-2000 Workshop on Fusion of Domain Knowledge with Data for Decision Support.*

Ngo, L. and Haddaway, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–171.

Riezler, S. (1998). *Probabilistic Constraint Logic Programming.* PhD thesis, Universität Tübingen. AIMS Report 5(1), 1999, IMS, Universität Stuttgart.