The Implementation and Application of a Logic-based Probabilistic Learning System

A proposal submitted to NSF 04-500: Information and Data Management, Unit: IIS

Neng-Fa Zhou
City University of New York

Summary

The past few years have witnessed a tremendous interest in logic-based probabilistic learning as testified by an increasing number of formalisms, systems, and applications. PRISM is a logic-based language that integrates logic programming, probabilistic reasoning, and EM learning. It allows for the description of independent probabilistic choices and their consequences in general logic programs. PRISM supports parameter learning, which amounts to training probabilistic models by using observed data. Our preliminary implementation of PRISM lacks scalability and is thus unable to sustain large applications. The goal of this project is to develop a high-performance PRISM system that can handle large sets of data. As tabling is the bottleneck in parameter learning, this project entails the development of an efficient tabling system. We will develop an efficient PRISM system that incorporates optimal tabling strategies, optimization techniques for tabling, and PRISM-specific optimization techniques for fast EM learning. We will also apply the implemented PRISM system to several problem domains including stochastic natural language processing, biological sequence analysis, game analysis, and heuristics learning for constraint solving, and use these applications to evaluate our system.

Intellectual merit: PRISM offers incomparable flexibility compared with specific statistical tools such as Hidden Markov Models (HMMs), Probabilistic Context Free Grammars (PCFGs) and discrete Bayesian networks. It is expected that the proposed implementation techniques will make PRISM competitive in performance as well with these specific tools. The proposed research will also have significant impact on the implementation of logic programming and deductive database systems.

Broader impact: This is a multidisciplinary research which will be conducted at CUNY with involvement of racially and ethnically diverse students, and also with collaboration of Taisuke Sato at Tokyo Institute of Technology. The proposed system will be made widely available to users inside and outside CUNY through the CUNY Software Institute.

The Implementation and Application of a Logic-based Probabilistic Learning System

A proposal submitted to NSF 04-500: Information and Data Management, Unit: IIS

Neng-Fa Zhou
City University of New York

1 Research Background and Objectives

The past few years have witnessed a tremendous interest in logic-based probabilistic learning as testified by large turnouts in related sessions in AI and data mining conferences, newly launched funding programs (e.g., the DARPA Evidence Extraction and Link Discovery program), and a number of formalisms and systems (e.g., BLP [13], CLP(BN) [7], PRM [14], PRISM [34], and SLP [20]). Logic-based probabilistic learning is a multidisciplinary research area that integrates relational or logic formalisms, probabilistic reasoning mechanisms, and machine learning and data mining principles. Logic-based probabilistic learning has found its way into many application areas including bioinformatics, diagnosis and troubleshooting, stochastic language processing, information retrieval, linkage analysis and discovery, robot control, and probabilistic constraint solving.

PRISM is a logic-based language that integrates logic programming, probabilistic reasoning, and EM learning [34]. It allows for the description of independent probabilistic choices and their consequences in general logic programs. PRISM supports parameter learning. For a given set of possibly incomplete observed data, PRISM can estimate the probability distributions to best explain the data. This power is suitable for applications such as learning parameters of stochastic grammars , training stochastic models for gene sequence analysis, user modeling, and obtaining probabilistic information for tuning systems performance. PRISM offers incomparable flexibility compared with specific statistical tools such as Hidden Markov Models (HMMs) [26], Probabilistic Context Free Grammars (PCFGs) [44] and discrete Bayesian networks [5, 23].

PRISM employs a proof-theoretic approach to learning. It conducts learning in two phases: The first phase searches for all the explanations for the observed data, and the second phase estimates the probability distributions by using the EM learning algorithm. Learning from flat explanations can be exponential in both space and time. To speed up learning, Sato and Kameya proposed learning from explanation graphs (called *graphical learning*) and using tabling to reduce redundancy in the construction of explanation graphs [34, 49]. Currently PRISM is implemented using our B-Prolog (www.probp.com), a constraint logic programming system that provides a tabling system called linear tabling [50, 48]. Tabling shares the same idea as dynamic programming in that both approaches make full use of intermediate results of computation. Using tabling in constructing explanation graphs resembles using dynamic programming in the Baum-Welch algorithm for the HMM [26] and the Inside-Outside algorithm for PCFG [1]. Although our system has had some success in learning from medium-size sets of data, it lacks scalability and is thus unable to handle large volumes of data. For example, a PDCG (Probabilistic Definite Clause Grammar) program takes hours to learn parameters from a corpus of 2000 sentences, and a game-modeling program can only analyze tens of game records in an hour.

The goal of this project is to develop a high-performance PRISM system that can handle large sets of data. As tabling is the bottleneck in parameter learning [34, 27, 30], this project entails the development of an efficient tabling system. Tabling to the evaluation of logic programs is as dynamic programming to problem solving. The main idea of tabling is to memorize the answers to some subgoals and use the answers to resolve subsequent variant subgoals [36, 42]. During the

last few years, several tabling systems have been implemented [32, 50, 12, 31, 48]. Some systems employ OLDT [36, 42] which relies on suspension and resumption of subgoals to compute fixpoints, and some other systems are based on our *linear tabling* mechanism [50, 48] which relies on iterative computation of looping subgoals to compute fixpoints.

Linear tabling is a framework from which different methods can be derived based on the strategies used in handling looping subgoals. The first objective of this project is to qualitatively and quantitatively analyze possible strategies and develop a method that is most efficient in terms of both space and time.

Current tabling systems incur considerable overhead to the execution of programs. Tamaki and Sato states the following in [36]: "The storage requirement [of tabling] can be too demanding in some cases and the overhead of table manipulation can be too large." This situation has not changed since. The second objective of this project is to develop optimization techniques to tackle this problem. In concrete, we will investigate the following optimization techniques: (1) *deferred tabling*, deferring tabling to reduce the overhead of over-tabled programs; (2) *semi-naive evaluation*, an algorithm employed in bottom-up evaluation of Datalog programs [39] to avoid redundant joins; and (3) *linear recursion optimization*, exploiting the characteristic of linear recursion in the evaluation of programs.

In PRISM, the EM learner walks a given explanation graph computing the inside and outside probabilities until the likelihood converges. Therefore, the structure of an explanation graph directly affects the performance of EM learning. We will design a compact representation for explanation graphs to facilitate EM learning.

We will also apply the implemented PRISM system to several different problem domains including stochastic natural language processing, biological sequence analysis, game analysis, and learning heuristics for constraint solving. These applications will on one hand be used to evaluate our implemented PRISM system and on the other hand be used to demonstrate the programming methodology of PRISM.

In summary, the objectives of this project include:

- Investigating different tabling strategies and designing an optimal method based on the results.
- Developing optimization techniques for the tabling method to eliminate redundant computations.
- Developing PRISM-specific optimization techniques to speed-up EM learning.
- Developing applications of PRISM to a number problems and using these applications to evaluate our PRISM system.

2 Prior Research Results on PRISM

There is a need for extending programming languages to facilitate the development of programs for *probabilistic computations* such as probabilistic modeling and learning. Various attempts have been made to meet this need [7, 14, 13, 20, 21, 24, 25, 30, 34, 37]. PRISM was first designed by Sato [33] who proposed a formal semantics, called *distribution semantics*, for logic programs with probabilistic built-ins, and derived an EM learning algorithm for the language from the semantics. PRISM was inspired by Poole's abduction language [25] which integrates Prolog with Bayesian networks. PRISM is known to subsume several other probabilistic logic languages such as SLP [20], PRM [14], and CLP(BN) [7].

PRISM is an extension of Prolog that provides built-ins for statistical modeling and learning. The following built-in predicates are provided:

- `msw(S,V)`: Describes an independent probabilistic choice where `S` is the name of a multiary random switch and `V` is the outcome of the switch. Each switch is defined by a sample space (i.e. a set of possible outcomes) and a probability distribution over the outcomes in the space. The probability distribution or parameters of the outcomes can be given by the programmer or learned from a set of observed facts. A decision process is modeled by a chain of independent choices each of which is represented as a call of `msw`.

- `sample(G)`: *Sample-executes* the program against the goal `G`. The execution is exactly the same as Prolog except for calls of `msw` which give different outcomes based on the probability distributions.

- `prob(G,P)`: Returns the probability `P` with which the goal `G` is true. It is assumed that all the probabilistic ground atoms in the body of each clause are probabilistically *independent* and the clauses defining a probabilistic predicate are probabilistically *exclusive*. With these assumptions, the probability of the conjunction `(A,B)` is computed as the product of the probabilities of `A` and `B` (*independent*), and the probability of the disjunction `(A;B)` is computed as the sum of the probabilities of `A` and `B` (*exclusive*). For a switch `msw(S,V)`, the probability is 1.0 if `V` is a variable, and the probability assigned to the outcome `V` if `V` is an element in the sample space. The probability of a goal can be computed from its *explanations*. An explanation of a goal is a conjunction of ground `msw` atoms that occur in a proof of the goal. PRISM provides a predicate called `probf(G,E)` for computing the explanations of a given goal.

- `learn(L)`: Estimates the parameters in the program to best explain the given list of observed facts `L`.

The following shows an illustrative example:

```
direction(D):-
    msw(coin,Face),
    (Face==head->D=left;D=right).

values(coin,[head,tail]).
```

The predicate `direction(D)` determines the direction to go by tossing a coin; D is bound to `left` if the head is shown, and to `right` if the tail is shown. The predicate `values(I,Space)` defines the sample space of a switch, where `I` is the identifier of a switch and `Space` is a list of possible outcomes of the switch. In general a decision process consists of a chain of many independent choices.

PRISM, as a symbolic statistical modeling language, subsumes several specific statistical tools such as the HMM [26], the PCFG [44] and the descrete Bayesian network [5, 23]. The following shows an interpreter for the HMM in PRISM.

```
hmm(L) :-
    msw(init,Si),
    hmm(Si,L).

hmm(S,[]).
hmm(S,[C|L]) :-
    msw(out(S),C),
    msw(tr(S),NextS),
    hmm(NextS,L).
```

For a given string, the choice `msw(init,Si)` determines the initial state, and for each state `S` the choice `msw(out(S),C)` determines the emitted symbol `C` and `msw(tr(S),NextS)` determines the transition from the state on the symbol. As can be understood from the decision process, two switches are used to represent the probability distributions of the symbol emissions and transitions.

PRISM employs a proof-theoretic approach to learning. It conducts learning in two phases: The first phase searches for all the explanations for the observed data, and the second phase estimates the probability distributions by using the EM learning algorithm.

An *explanation* for an observed fact is a set of switches that occur in a proof of the fact. Let I be the set of switches, and V_i be the sample space of switch i. For each switch $msw(i, v)$, $\theta_{i,v}$ denotes the probability of the outcome v in V_i. The following assertion ensures that each switch has a probability distribution

$$\forall_{i \in I} \Sigma_{v \in V i} \theta_{i,v} = 1.0.$$

Let F be a set of observed facts. For each fact $f \in F$, E_f denotes the set of explanations. Let $e \in E_f$ be an explanation. The probability of e is the product of the probabilities of all the switches in the explanation:

$$\theta_e = \prod_{msw(i,v) \in e} (\theta_{i,v})$$

The probability of fact f is the sum of the probabilities of all its explanations:

$$\theta_f = \sum_{e \in E_f} (\theta_e)$$

The *log likelihood* of fact f is defined as $\ln(\theta_f)$. For each explanation $e \in E_f$, let $\delta_{i,v}(e)$ denote the number of occurrences of the switch $msw(i, v)$ in e. Figure 1 shows the EM algorithm. The algorithm repeats the estimation until the likelihood of the observed facts becomes stable.

The use of the term $\eta_{i,v}$, which estimates the number of occurrences of the switch $msw(i, v)$ that contribute to the observed facts, is essential in the algorithm. The probability of $msw(i, v)$ is estimated as the ratio of its count to the count of all the outcomes of the switch.

$$\theta_{i,v} = \frac{\eta_{i,v}}{\sum_{v' \in V i} (\eta_{i,v'})}$$

Learning based on plain explanations can be exponential. Sato and Kameya proposed a graphical EM learning algorithm in which tabling is used to construct explanation graphs [34], and we have implemented the algorithm by using the native tabling mechanism in B-Prolog [49]. It has been proved that PRISM not only subsumes the symbolic-statistical models such as the HMM [26, 26], the PCFG [44, 18]) and the Bayesian networks [23, 5], but also enjoy the same time complexity as the native EM algorithm used in each of these models provided that the graphical EM algorithm and tabling are used [34].

3 Prior Research Results on Tabling

The SLD resolution used in Prolog may not be complete or efficient for programs in the presence of recursion. For example, for a recursive definition of the transitive closure of a relation, a query may never terminate under SLD resolution if the program contains left-recursion or the graph represented by the relation contains cycles even if no rule is left-recursive. For a natural definition of the Fibonacci function, the evaluation of a subgoal under SLD resolution spawns an exponential number of

procedure em(F) **begin**
initialize ϵ to a small positive number;
foreach $i \in I$, $v \in V_i$ initialize $\theta_{i,v}$;
$\lambda^1 = \sum_{f \in F}(\ln(\theta_f))$; /* initial likelihood */
repeat
$\lambda^0 = \lambda^1$;
foreach $i \in I$, $v \in V_i$
$\eta_{i,v} = \sum_{f \in F}(\frac{\sum_{e \in E_f}(\theta_e \times \delta_{i,v}(e))}{\theta_f})$
/* expected count of $msw(i,v)$ */
foreach $i \in I$, $v \in V_i$
$\theta_{i,v} = \frac{\eta_{i,v}}{\sum_{v' \in V_i}(\eta_{i,v'})}$
$\lambda^1 = \sum_{f \in F}(\ln(\theta_f))$;
until $\lambda^1 - \lambda^0 < \epsilon$
end

Figure 1: The EM algorithm

subgoals, many of which are variants. The lack of completeness and efficiency in evaluating recursive programs is problematic: novice programmers may lose confidence in writing declarative programs that terminate and real programmers have to reformulate a natural and declarative formulation to avoid these problems, resulting in less declarative and less readable programs.

Tabling [36, 42] is a technique that can get rid of infinite loops for bounded-term-size programs and redundant computations in the execution of recursive Prolog programs. The main idea of tabling is to memorize the answers to some subgoals and use the answers to resolve subsequent variant subgoals. This idea of caching previously calculated solutions, called *memoization*, was first used to speed up evaluation of functions [19].

Tabling in Prolog not only is useful in the problem domains that motivated its birth, such as program analysis, parsing, deductive database and theorem proving but also has been found essential in several other problem domains such as model checking [28] and probabilistic logic learning [27, 34, 49].

OLDT [36] is the first revised SLD resolution that accommodates the idea of tabling. In OLDT, a table area is used to record subgoals and their answers. When a subgoal (*producer*) is first encountered in execution, it is resolved by using program clauses just as in SLD resolution with the exception that the subgoal and its answers are recorded in the table. When a subgoal (*consumer*) is encountered that is subsumed by one of its ancestors, OLDT does not expand it as in SLD resolution but rather uses the answers in the table to resolve it. After the answers are exhausted, the computation of the consumer is suspended until the producer produces new answers into the table. The process is continued until the fixpoint is reached, i.e. when no answers are available for consumers and no producers can produce any new answers. Several extensions of OLDT including SLG [6] and SLS [4] have been developed for evaluating general logic programs with negation. XSB is the first Prolog system that successfully supports tabling [32].

OLDT is non-linear in the sense that the state of a consumer must be preserved before execution backtracks to its producer. This non-linearity requires freezing stack segments [32] or copying stack segments into a different area [8] before backtracking takes place. Recently, another formalism,

called linear tabling[1], has emerged as an alternative tabling method [35, 50, 12]. The main idea of linear tabling is to use iterative computation rather than suspension to compute fixpoints. A significant difference between linear tabling and OLDT lies in the handling of variant descendents of a subgoal. In linear tabling, after a descendent consumes all the answers, it either fails or turns into a producer, producing answers by using the alternative clauses of the ancestor. A subgoal is called a *looping subgoal* if a variant occurs as a descendent in its evaluation. The evaluation of looping subgoals must be iterated to ensure the completeness of evaluation.

Linear tabling is a framework from which different methods can be derived based on the strategies used in handling looping subgoals in forward execution, backtracking, and iteration. The framework can be defined by three primitives on tabled subgoals: *start*(A), *memo*(A), and *check_completion*(A).

- *start*(A) This primitive is executed when a tabled subgoal A is encountered. The subgoal A is registered into the table if it is not registered yet. If A's state is *complete* meaning that A has been completely evaluated before, then A is resolved by using the answers in the table.

  If A is a pioneer of the current path, meaning that it is encountered for the first time, then it is resolved by using program clauses.

  If A is a follower of some ancestor A_0, meaning that a loop has been encountered, then it is first resolved by using the answers in the table. After the answers are exhausted, two possible actions can be taken: One is to fail A. This strategy is called *FDF* (Fail Descendent Followers). The other is to resolve A by using the alternative clauses of its ancestor A_0. This strategy is called *SAC* (Steal Alternative Clauses).

- *memo*(A) This primitive is executed when an answer is found for the tabled subgoal A. If the answer A is already in the table, then fail; otherwise the answer is added into the table. Two different strategies can be used after the answer is added. One is called *Consume Immediately After Produce* (CIAP), which lets the subgoal succeed and consume the answer. The other strategy is called *Produce All Before Consume* (PABC), which fails the primitive and forces the system to backtrack to find the next answer.

- *table_completion*(A) This primitive is executed when the subgoal A is being resolved by using program clauses and all the clauses have been tried. If A has never occurred in a loop, then A's state can be set to *complete* and A can be failed after all the answers are consumed.

  If A is a top-most looping subgoal, then different actions should be taken depending on whether this is A's normal evaluation or re-evaluation. These actions depend again on what strategies are used by *start*(A) and *memo*(A). In any case, if no new answer is produced in the last round of re-evaluation of A, then A's state can be set to *complete* and A can be failed after all the answers are consumed.

  For a pioneer in a loop that is not a top-most looping subgoal, two different strategies can be used after the subgoal exhausts all the clauses and answers. One is to fail the subgoal. This means that under this strategy re-evaluation starts only at top-most looping subgoals. The other strategy is to iterate the evaluation of the subgoal until it reaches a temporary fixpoint. The fixpoint is temporary since the top-most subgoal on which this subgoal depends has not been completely evaluated.

Linear tabling is relatively easy to implement on top of a WAM-like abstract machine thanks to its linearity. Linear tabling is more space efficient than suspension-based methods since the states of

[1] Notice that the word *linear* here has nothing to do with complexity or linear logic. It has the same meaning as `L` in SLD: a derivation is made up of a sequence of goals $G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_k$ such that G_{i+1} is derived from G_i.

subgoals need not be preserved. With the *subgoal optimization* technique proposed in [48], linear tabling has been shown to be as competitive as OLDT in terms of time efficiency as well.

4 Proposed Research

As described above, linear tabling is a framework from which different methods can be derived: each is a combination of strategies for handling looping subgoals in forward execution, backtracking, and iteration. For example, for the forward execution of a looping subgoal that is a descendent of a variant ancestor, one strategy is to fail the subgoal after it consumes all the available answers, and another strategy is to let the descendent produce answers by using the alternative clauses of the ancestor. The decisions on how to handle looping subgoals in forward execution, backtracking, and iteration are orthogonal, and different methods can be designed by combining different strategies. One of the objectives of this project is to qualitatively and quantitatively analyze possible strategies and perfect a method that is most efficient in terms of both space and time.

No tabling method could lead to an efficient system without optimization. Another objective of this project is to investigate several optimization techniques. We will also be investigating the method for handling cuts under various kinds of strategies.

4.0.1 Deferred tabling

Current tabling systems incur significant overhead to the execution of programs. Tamaki and Sato states the following in [36]: "The storage requirement [of tabling] can be too demanding in some cases and the overhead of table manipulation can be too large." This situation has not much changed since. For example, for the naive reverse program both XSB and B-Prolog slow down by over 30 times if the predicates are declared tabled.

The overhead problem of tabling is a common problem to all current tabling systems. We propose a technique, called *deferred tabling* to tackle this problem.

In current tabling systems, a table-declared subgoal is tabled the first time when it is encountered regardless of whether the answers can be reused or not. For a predicate for which no subgoal occurs twice, tabling it is a waste. For a predicate for which even if some subgoals may occur multiple times, most subgoals may occur only once. Tabling those single-occurring subgoals is a waste. This technique amounts to defer tabling a subgoal until a sufficiently close subgoal occurs again in execution.

The deferred tabling tachnique is analogous to the buffering technique used in operating systems. One of the key components in the design of this technique is to define the closeness of subgoals. We will rely on subgoal abstraction to check the closeness of subgoals and explore several definitions for subgoal abstraction.

4.0.2 Semi-naive evaluation in linear tabling

Linear tabling relies on iterative evaluation of top-most looping subgoals to compute fixpoints. Blind re-computation of all subgoals and clauses is not computationally acceptable. A system should re-evaluate only those subgoals and should use only those clauses and answers that can contribute to the generation of new answers.

The semi-naive algorithm used in bottom-up evaluation for Datalog programs [2, 39] can be used to avoid redundant joins in linear tabling. In concrete, in each round of evaluation, the join of the subgoals in the body of each rule must involve at least one new answer produced in the previous

round. Let "$H{:-}A_1, ..., A_k, ..., A_n$" be a clause where A_k is the last subgoal in the body that may depend on H (i.e., all the subgoals to the right of A_k that have occurred in early rounds must be already complete when H is re-evaluated). For each combination of the answers of A_1, $\cdots$, and A_{k-1}, if the combination does not contain any new answers then A_k should consume new answers only.

We need to surmount several hurdles in order to implement the semi-naive algorithm in linear tabling. Firstly, we need to identify the last subgoal A_k in the body of each rule that depends on the head. We can extract this kind of information from the call-graph of the program or the dependence relation of all possible subgoals that can be reached from a query. Secondly, for each combination of answers for the subgoals of A_1, $\cdots$, and A_{k-1}, we need to detect whether there is any new answer involved. This detection is not as easy as it looks like due to the fact that during the evaluation of a top-most looping subgoal T anther subgoal T' (which is an ancestor of T) can be discovered to be a dependent subgoal of T and thus the next round of evaluation will start at T' not T. The difficulty is caused by the fact that an answer of a subgoal old to T can be new to T'.

The semi-naive algorithm will have as great an impact on linear tabling as it has on bottom-up evaluation. With this algorithm, linear tabling can constantly beat the magic-sets method [39] since instantiation information of goals are passed down automatically in top-down evaluation and linear tabling enjoys far better space efficiency than the OLDT and bottom-up evaluation methods.

4.0.3 Optimization of linear recursion

A predicate is said to be *linear-recursive* if it contains only one recursive clause which takes the form:

$$p(\ldots) :- A_1, ..., A_i, p(\ldots), A_{i+1}, ..., A_n$$

where $p(\ldots)$ is the only recursive subgoal depending on p in the body. In particular, the predicate is said to be *left recursive* if $p(\ldots)$ is the first subgoal, *right recursive* if $p(\ldots)$ is the last subgoal, and *inner recursive* if $p(\ldots)$ is neither first nor last subgoal in the body. Many relations in deductive database such as *transitive closure* and *same generation* can be defined by using linear recursion (see [39] for related references). The HMM interpreter shown before is also linear recursive.

For linear recursion, no two loops can intermingle with each other. This means that the latest looping subgoal must be top-most looping subgoal. This characteristic can be exploited to optimize further the evaluation algorithm.

4.0.4 PRISM-specific optimization techniques

The graphical EM learning algorithm walks a given graph many times until it converges. Therefore, a compact and efficient representation for explanation graphs is crucially important. To this end, we propose two techniques, namely *auto-tabling* and *graph compression*.

The auto-tabling technique amounts to devising a standard for automatically tabling predicates. It is daunting for programmers to declare what predicates need to be tabled. Users' declarations can be inaccurate. Programs may slow down significantly because of over-tabling or may fall into infinite loops because some predicates are not tabled. Furthermore, table declarations affect directly the structure of the resulting explanation graph. Therefore, such a standard should take the following into consideration: (1) the overhead of over-tabling, (2) the cost of re-computation because of lack of tabling; (3) the possibility of infinite loops; and (4) the compactness of the resulting explanation graph.

The resulting explanation graph from a program may not be optimal even if a wise standard is used in selecting table predicates. The graph compression technique reduces the size of the graph further by eliminating redundant nodes and edges, and by combing possible subgraphs.

4.1 Applications of PRISM

PRISM is suited for building complex systems that involve both symbolic and probabilistic elements such as discrete hidden Markov models, stochastic string/graph grammars, game analysis, data mining, performance tuning and bio-sequence analysis.

4.1.1 Stochastic natural language processing

The difficulty of natural language processing (NLP) is due to the ambiguity of languages. For a grammar that describes a nontrivial subset of a natural language, each sentence may receive an overwhelmingly large number of parse trees. It is of paramount importance for NLP systems to identify most plausible parse trees automatically. During the last few years, several formalisms of stochastic grammars have been developed for this purpose [18].

PCFGs [44, 18] are a probabilistic extension of context-free grammars (CFGs) where each nonterminal is associated with a probability distribution over the productions. In a language defined by a PCFG, each derivation receives a probability and each parse tree receives the probability of its corresponding derivation. Derivation steps are assumed to be independent.

The encoding of a PCFG in PRISM is quite straightforward. For each nonterminal we use a switch that has the nonterminal as the name and an outcome for each of the productions. The outcome corresponding to each production is encoded as the string of symbols on the right hand side of the production. The following shows a parser for a PCFG:

```
parse([Wd|R],[Wd|L0],L):- % matching
   terminal(Wd),!,
   parse(R,L0,L).
parse([A|R],L0,L2):-      % reduction
   msw(A,B),              % choose a production
   parse(B,L0,L1),
   parse(R,L1,L2).
pdcg([],L,L).
```

Suppose `s` is the start symbol and `S` is a given string encoded as a list of words, the goal `parse([s],S,[])` tries to generate a parse tree for `S`. This program can be used not only to find most plausible parse trees for a given sentence but also to estimate the probability distributions from a corpus.

We have encoded the ATR grammar [40] in PRISM and succeeded in training the grammar with a corpus of 2000 sentences [49]. We will experiment with larger corpora and try other grammars developed by the NLP community.

4.1.2 Biological sequence analysis

Many problems in computational biology can be reduced to some sort of linear sequence analysis. Since a family of sequences can only be described statistically, traditional string matching algorithms are not useful. The Hidden Markov Model, which is originally developed for speech recognition [26], is very well suited for many tasks in biological sequence analysis [43, 9].

An HMM is a probabilistic finite automaton where all the transitions and symbol emissions are probabilistic [26]. Various kinds of HMMs have been designed for biological sequence analysis [10]. It is straightforward to implement HMMs in PRISM as illustrated by the example shown before. For each state, we use two switches to describe the probability distributions of the transitions and symbol emissions, respectively. With this representation of an HMM, we can write programs to carry out tasks such as aligning and scoring sequences. It is also possible to train the model, i.e. estimate the probability distributions from observed sequences.

We will translate the Pfam protein database [3] into PRISM and write programs for training models and analyzing biological sequences. We will compare our EM learner in PRISM with the Baum-Welch algorithm [26] for model training. An HMM can be considered a probabilistic automaton for a probabilistic regular grammar. In contrast, PRISM is based on Horn logic, which is far more powerful than regular grammars. Certain problems such as RNA folding and structure predication for proteins [15] require a more sophisticated grammar than a regular grammar. We will apply PRISM to these problems and compare it with other approaches developed in computational biology.

4.1.3 Bayesian networks

Bayesian networks have become a popular representation for encoding and reasoning about uncertainty in various applications [5, 23]. A Bayesian network is a graphical model for probabilistic relationships among a set of random variables. PRISM subsumes Bayesian networks. The conditional probability table in a Bayesian network can be represented by *complex* switches in PRISM. Let B and C be two random variables. Assume C has the possible outcomes $\{c_1, ..., c_n\}$. The conditional event $(B|C)$ can be represented by using n switches: $msw(b(c_i), V_i)$ (i=1,...,n). The graphical EM learning algorithm can be used to train parameters of Bayesian networks from data.

The XML-BNIF[2], which was developed by Microsoft, has emerged as a standard format for researchers to share Bayesian network models. We will develop a converter in Prolog that converts XML-BNIF to and from PRISM. We will also compare our PRISM system with the specific modeling and inference tools for Bayesian networks such as the tool developed by Hugin.

4.1.4 Learning heuristics for constraint solving

Many search problems can be formulated as constraint satisfaction problems (CSPs) [16, 38]. A CSP is defined by a set of variables each of which is defined over a domain and a set of constraints amongst the variables. A solution to a CSP is an assignment of a value to each variable such that all the constraints are satisfied. We have developed a language called *Action Rules* [46] and implemented several propagation-based constraint solvers using the system. We have also applied the solvers to problems ranging from packing [47], VLSI design [45], to planning.

Most CSPs are NP-hard problems for which we have to rely on good heuristics to search for solutions. For CSPs, the strategies used to order variables and values can have a dramatic effect on the performance. Various kinds of strategies have been proposed (e.g., the *first-fail principle*

[2] http://research.microsoft.com/dtas/bnformat/

for ordering variables [16] and the *mini-conflict* strategy for ordering values [11]). Nevertheless, for many problems users still have to experiment with different strategies and tune them manually. This process can be very tedious and painful.

We propose to integrate PRISM into our constraint solvers so that strategies for ordering variables and values can be learned automatically. More specifically, we use a switch for variable selection and use a switch for value selection for each variable. Learning can be conducted from traces, which can be obtained from runs of the program on small problems or partial runs on large problems. Learning can be done offline or can be intermingled with the search process. We plan to apply the integrated system to several real-world problems including packing, VLSI design, scheduling and planning problems.

5 Experience and Capabilities of the PI

The PI, Dr. Neng-Fa Zhou, has been an active researcher in programming language systems for over ten years. He has authored over thirty papers on programming language, constraint-solving, graphics, and machine learning systems including over fifteen papers published in top journals (ACM TOPLAS, Journal of Logic Programming, Theory and Practice of Logic Programming, Journal of Functional and Logic Programming, and Software Practice and Experience) and major conferences. His papers on compilation of logic programs, constraint solving, and tabling have received a number of citations. He is the main developer of the B-Prolog system, a cutting-edge constraint logic programming system which has tens of thousands users worldwide in both academia and industry.

6 Plan of Activities

The implementation and application of PRISM are the two objectives of this research. A preliminary version of PRISM has been developed. It is therefore possible and desirable to conduct the research activities in parallel. This project will be conducted jointly by the PI's research group at CUNY and Taisuke Sato's group at Tokyo Institute of Technology. The responsibility of PI's research group includes the development of an efficient PRISM system and the applications of PRISM to biological sequence analysis and heuristics learning for constraint solving.

We expect that three PhD and several master and undergraduate students will be involved in this project. Funds are requested from NSF for supporting the PhD students. Separate funding is being sought from the CUNY software institute for supporting master and undergraduate students.

6.1 Year I

The major activity for the PI and part of the students in the first academic year will be the study of the implementation techniques for PRISM. As tabling plays a prominent role in constructing explanation graphs, this project entails the development of an efficient tabling system. Several different strategies can be used in different stages of tabling. Since most of the strategies are orthogonal, a dozen combinations of the strategies are possible. If optimization techniques and data structures for tabled subgoals and answers are taken into account, the number of combinations can be very large. We will first analyze possible methods qualitatively and identify those that are promising. We will then design data structures and optimization techniques for those methods. The soundness and completeness of the optimization techniques will be proved.

Another activity to be undertaken in the first academic year is to apply the current version of PRISM to bio-sequence analysis and heuristics learning for constraint solving. Attempts will be made to convert existing HMMs for bio-sequence analysis to PRISM and train the models using small size data sets. As for heuristics learning for constraint solving, we will try to model the ordering of variables and values as a random decision process and analyze the model by using different constraint satisfaction problems such as SAT and coloring problems.

6.2 Year II

By the beginning of Year II we will have the first prototype of a new PRISM system ready. The primary research task of this year will be the implementation and evaluation of the optimization techniques. The prototype applications developed in the first year and the applications developed by Sato Taisuke's group will be used to evaluate the implementation.

With the maturity of the PRISM system, the applications will be scaled up to handle larger sets of data. As long as the application to bio-sequence analysis is concerned, problems that are considered difficult to solve with regular HMMs, such as RNA folding and structure predication for proteins, will be studied. As for the application to heuristics learning, real-world problems such as packing, VLSI design, and scheduling will be used to train and evaluate the models.

During this year, all the participants in the project will attend at least one professional conference and present research results.

6.3 Year III

The focus of this year is on the application, evaluation, and dissemination of results. The PI and students will make refinements to the system which they determine as necessary. The results will be disseminated through several avenues including news groups, conferences, demos, and possibly journals. The PI teaches at the CUNY Graduate Center a course on "Programming Languages and Their Implementation" where the implementation techniques of some major high-level languages are covered. Two or three lectures will be planed to cover the implementation techniques of Constraint Logic Programming languages including tabling.

7 Intellectual Merit of Proposed Research

Probabilistic models are becoming increasingly important in analyzing large volumes of data in bioinformatics, diagnosis and troubleshooting, stochastic language processing, information retrieval, linkage analysis and discovery, user modeling, and optimization of responsiveness of various kinds of hardware and software systems. Because of the usefulness of probabilistic models and the potential of huge markets for the applications, probabilistic modeling has received tremendous attention among academia and industry as well. Numerous software packages have been developed during the last decade[3], companies like Microsoft have started a number of research projects in related areas.

Currently, different specialized tools are used for different applications. For example, hidden Markov models are used for analyzing biological sequences [43], the Viterbi [41] algorithm is used for searching for most probable parses for ambiguous languages, the Bayesian networks are used

[3]See http://fulcrum.physbio.mssm.edu/ sdy/panningsearch/HiddenMarkovModels.htm for a list of HMMs for gene sequence analysis, and http://www.ai.mit.edu/ murphyk/Bayes/bnsoft.html for a list of software packages for Bayesian networks.

in diagnosis [23]. As a probabilistic modeling tool, PRISM subsumes all the popular tools such as the HMM, the PCFG, and the discrete Bayesian networks. As a symbolic language, PRISM offers incomparable flexibility by allowing the use of arbitrary logic programs to describe probability distributions and reason about the consequences of choices. The graphical EM learning algorithm renders the more specific EM learning algorithms such as the Baum-Welch [26] and Inside-Outside [1] algorithms unnecessary. Moreover, an implementation of graphical learning based on a fast tabling system can give comparable efficiency with those specialized EM learning algorithms. Several other logic-based probabilistic modeling and learning systems (e.g., BLP [13], CLP(BN) [7], PRM [14], and SLP [20]) have been developed during the last decade. Compared with these systems, PRISM enjoys both generality and efficiency. It is the first system that seamlessly integrates EM learning with a tabling system.

Our research will also have a significant impact on logic programming and deductive database research. As far as logic programming is concerned, the need to extend Prolog to narrow the gap between declarative and procedural readings of programs has been previously urged [22]. Although tabling has been perceived as necessary to narrow the gap, it has not become a standard feature. One primary reason may be the lack of an easy-to-implement and efficient tabling method. Our optimized tabling method will accelerate the acceptance of tabling as a standard feature of logic programming. Deductive database used to be a very hot research area. A number of bottom-up evaluation methods have been invented for evaluating recursive Datalog rules [2, 39], and several systems have been developed [29, 17]. Deductive database was expected to replace relational database as the next generation database model. Nevertheless, deductive database research failed to meet the early expectations. The reasons for this failure are that bottom-up evaluation lacks the efficiency and flexibility of top-down evaluation and most Datalog implementations do not allow function symbols which are necessary for representing complex data in many application domains. Our linear tabling method has the potential to revitalize the research and development of deductive database systems and applications.

8 Broader Impact of Proposed Research

Our research will have broader impact on society and education. The Computer Science PhD program at CUNY is the largest in New York City, which enrolls a body of 130 racially and ethnically diverse students. The founding of the CUNY Software Institute (CISDD) (www.cisdd.org) and the hiring of distinguished scholars such as Sergei Artemov, Robert Haralick, and Amotz Bar-Noy have significantly strengthened the program. Our research will seek to involve and develop the academic skills of CUNY students including undergraduate and minority students.

Our research will be conducted collaboratively by groups at CUNY and Tokyo Institute of Technology. It will therefore foster collaboration between these two institutions. The proposed system will be directly applicable to many other projects that are being undertaken at CUNY such as Simon Parsons's multi-agent systems project, Susan Epstein's constraint programming project, and Rohit Parikh's research in game and voting theories.

Our research results will be disseminated through several channels including courses, seminars, conferences, and journals. The PI will integrate part of the results developed by this project into his courses taught at the CUNY graduate center and other CUNY colleges. The software system developed by this project will be made available to the public.

The CISDD serves as a hub that connects software industries and institutions in the greater New York region. We will make use of office spaces for students and visitors provided by CISDD and establish a close relationship with industry through CISDD.

References

[1] Baker, J. K. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America* (1979), pp. 547–550.

[2] Bancilhon, F., and Ramakrishnan, R. An amateur's introduction to recursive query processing strategies. *Proc. of ACM SIGMOD '86* (1986), 16–52.

[3] Bateman, A., Birney, E., Durbin, R., Eddy, S. R., Howe, K. L., and Sonnhammer, E. L. The pfam protein families database. *Nucleic Acids Res. 28* (2000), 263–266.

[4] Bol, R. N., and Degerstedt, L. Tabulated resolution for the well-founded semantics. *Journal of Logic Programming 34*, 2 (1998), 67–109.

[5] Castillo, E., Gutierrez, J. M., and Hadi, A. S. *Expert Systems and Probabilistic Network Models*, 1 ed. Springer, 1997.

[6] Chen, W., and Warren, D. S. Tabled evaluation with delaying for general logic programs. *Journal of the ACM 43*, 1 (1996), 20–74.

[7] Costa, V. S., Page, D., Qazi, M., and Cussens, J. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of 2003 Conference on Uncertainty in Artificial Intelligence (UAI-03)* (2003), Morgan Kaufmann Publishers.

[8] Demoen, B., and Sagonas, K. CHAT: The copy-hybrid approach to tabling. In *Proceedings of Practical Aspects of Declarative Programming (PADL)* (1999), LNCS 1551, Springer-Verlag, pp. 106–121.

[9] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. *Biological sequence analysis*. Cambridge University Press, 1998. ISBN: 0–521–62971–3.

[10] Eddy, S. R. Profile hidden markov models (review). *Bioinformatics 14*, 9 (1998), 755–763.

[11] Frost, D., and Dechter, R. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'95* (Aug. 1995), pp. 572–578.

[12] Guo, H.-F., and Gupta, G. A simple scheme for implementing tabled logic programming systems based on dynamic reordering of alternatives. In *Proceedings International Conference on Logic Programming (ICLP)* (2001), LNCS 2237, Springer-Verlag, pp. 181–195.

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

[14] Koller, D. Probabilistic relational models. In *Proceedings of the 9th International Workshop on Inductive Logic Programming* (1999), S. Džeroski and P. Flach, Eds., vol. 1634 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 3–13. Invited paper.

[15] Krogh, A. An introduction to hidden markov models for biological sequences. In *Computational Methods in Molecular Biology* (Amsterdam, 1998), S. L. Salzberg, D. B. Searls, and S. Kasif, Eds., Elsevier, pp. 45–63.

[16] Kumar, V. Algorithms for constraint satisfaction problems: A survey. *AI Magazine 13* (1992), 32–44.

[17] Liu, M. Deductive database languages: Problems and solutions. *ACM Computing Surveys 31*, 1 (1999), 27–62.

[18] Manning, C. D., and Schütze, H. *Foundations of Statistical Natural Language Processing*. The MIT Press, 2000.

[19] Michie, D. "memo" functions and machine learning. *Nature* (1968), 19–22.

[20] Muggleton, S. Stochastic logic programs. In *Advances in Inductive Logic Programming*, L. D. Raedt, Ed. IOS Press, 1996, pp. 254–264.

[21] Park, S. A calculus for probabilistic languages. In *Proceedings of the 2003 ACM SIGPLAN international workshop on Types in languages design and implementation* (2003), ACM Press, pp. 38–49.

[22] Parker, D., Carey, M., Jarke, M., Sciore, E., and Walker, A. Logic programming and databases. In *Expert Database Systems* (1986).

[23] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, 1987.

[24] Pfeffer, A., Koller, D., Milch, B., and Takusagawa, K. T. Spook: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)* (1999), K. B. Laskey and H. Prade, Eds., Morgan Kaufmann Publishers, pp. 541–550.

[25] Poole, D. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64*, 1 (1993), 81–129.

[26] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recoginition. *Proceedings of the IEEE 77* (1989), 257–286.

[27] Raedt, L. D., and Kersting, K. Probabilistic logic learning. *SIGKDD Explorations 5* (2003), 31–48.

[28] Ramakrishnan, C. Model checking with tabled logic programming. In *ALP News Letter* (2002), ALP.

[29] Ramakrishnan, R., and Ullman, J. D. A survey of deductive database systems. *Journal of Logic Programming 23*, 2 (May 1995), 125–149.

[30] Ramsey, N., and Pfeffer, A. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2002), ACM Press, pp. 154–165.

[31] Rocha, R., Silva, F., and Costa, V. S. On a tabling engine that can exploit or-parallelism. In *Proceedings International Conference on Logic Programming (ICLP)* (2001), LNCS 2237, Springer-Verlag, pp. 43–58.

[32] Sagonas, K., and Swift, T. An abstract machine for tabled execution of fixed-order stratified logic programs. *ACM Transactions on Programming Languages and Systems 20*, 3 (1998), 586–634.

[33] Sato, T. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming* (1995), L. Sterling, Ed., MIT Press, pp. 715–730.

[34] Sato, T., and Kameya, Y. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* (2001), 391–454.

[35] Shen, Y.-D., Yuan, L., You, J., and Zhou, N.-F. Linear tabulated resolution based on Prolog control strategy. *Theory and Practice of Logic Programming (TPLP) 1*, 1 (2001), 71–103.

[36] Tamaki, H., and Sato, T. OLD resolution with tabulation. In *Proceedings of the Third International Conference on Logic Programming* (1986), E. Shapiro, Ed., LNCS, Springer-Verlag, pp. 84–98.

[37] Thrun, S. Probabilistic robotics. *Communications of the ACM 45*, 3 (Mar. 2002), 52–57.

[38] Tsang, E. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[39] Ullman, J. D. *Database and Knowledge-Base Systems*, vol. 1 & 2. Computer Science Press, 1988.

[40] Uratani, N., Takezawa, T., Matsuo, H., and Morita, C. ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, 1994. In Japanese.

[41] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory IT-13* (Apr. 1967), 260–269.

[42] Warren, D. S. Memoing for logic programs. *Comm. of the ACM, Special Section on Logic Programming 35*, 3 (1992), 93.

[43] Waterman, M. S. *Introduction to Computational Biology: Maps, Sequences and Genomes*. CRC Press, 1995.

[44] Wetherell, C. S. Probabilistic languages: A review and some open questions. *ACM Computing Surveys 12*, 4 (1980), 361–379.

[45] Zhou, N.-F. A logic programming approach to channel routing. In *Proceedings of the 12th International Conference on Logic Programming* (June 13–18 1995), L. Sterling, Ed., MIT Press, pp. 217–232.

[46] Zhou, N.-F. Implementing constraint solvers in B-Prolog. In *IFIP World Congress, Intelligent Information Processing*. Kluwer Academic Publishers, 2002, pp. 249–260.

[47] Zhou, N.-F. CGLIB — a constraint-based graphics library. *Software Practice and Experience 33*, 13 (Nov. 2003), 1199–1216.

[48] Zhou, N.-F., and Sato, T. Efficient fixpoint computation in linear tabling. In *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming* (2003), pp. 275–283.

[49] Zhou, N.-F., Sato, T., and Hasida, K. Toward a high-performance system for symbolic and statistical modeling. In *IJCAI Workshop on Learning Statistical Models from Relational Data* (2003), pp. 153–159.

[50] Zhou, N.-F., Shen, Y.-D., Yuan, L., and You, J. Implementation of a linear tabling mechanism. *Journal of Functional and Logic Programming 2001(1)* (2001), 1–15.

Budget Justification

A total of $443,792 is requested for our three-year project. In this section, we provide detailed justification for each expenditure.

Senior personnel ($20,000 each year)

Zhou will be spending two months in each summer of the three years working on the project. A one-month summer salary of $8,000 is budgeted for him. The amount is equivalent to 1/9th of his current nine-month salary. His work for the second month will be supported by other sources.

Zhou will be spending significant amounts of time on the project during the academic year. Release time is requested for one course at the rate of $6,000 a course for each semester. The teaching load at CUNY is 21 credits a year for tenured faculty, and therefore the requested amount of release time is necessary.

Graduate students ($54,000 for first year, and 4% increase for years 2 and 3)

Three PhD students will participate in this project. Each student will receive $18,000 for 950 hours worth of work in the first year, and an estimated 4% increase from the previous year for the subsequent years. This project will also involve master and undergraduate students. Separate funding is being requested for supporting these students from the CUNY software institute.

Fringe benefits

The fringe benefit rate for summer salary is 20% and that for release time is 28% for senior personnel. The fringe benefit rate for graduate students is 8%.

Equipment

No funds are requested for equipment. CUNY (the graduate center, the software institute, and Brooklyn College) will provide facilities and equipment for this project.

Travel ($6,000 each year)

Funds are requested for domestic and international travel in each of the three years to allow the project senior personnel to present results at appropriate conferences and symposia on Data Mining, Logic Programming, AI, and Database Management. Funds are also requested for the graduate students supported by the project to participate and present results in domestic conferences.

Indirect costs

The indirect costs are computed at 53% of the direct costs.

Other

Funds are requested in the first year for purchase of a desktop personal computer to be used for the project. The computer will have a 64-bit CPU and run 64-bit Windows and Linux. It will also have large memory and disk capacity that is needed to meet our data requirements. Funds are requested for supplies, software, books, and proceedings. A $500 honorarium is requested for visitors who give lectures at CUNY.

References

[1] Baker, J. K. Trainable grammars for speech recognition. In *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America* (1979), pp. 547–550.

[2] Bancilhon, F., and Ramakrishnan, R. An amateur's introduction to recursive query processing strategies. *Proc. of ACM SIGMOD '86* (1986), 16–52.

[3] Bateman, A., Birney, E., Durbin, R., Eddy, S. R., Howe, K. L., and Sonnhammer, E. L. The pfam protein families database. *Nucleic Acids Res. 28* (2000), 263–266.

[4] Bol, R. N., and Degerstedt, L. Tabulated resolution for the well-founded semantics. *Journal of Logic Programming 34*, 2 (1998), 67–109.

[5] Castillo, E., Gutierrez, J. M., and Hadi, A. S. *Expert Systems and Probabilistic Network Models*, 1 ed. Springer, 1997.

[6] Chen, W., and Warren, D. S. Tabled evaluation with delaying for general logic programs. *Journal of the ACM 43*, 1 (1996), 20–74.

[7] Costa, V. S., Page, D., Qazi, M., and Cussens, J. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of 2003 Conference on Uncertainty in Artificial Intelligence (UAI-03)* (2003), Morgan Kaufmann Publishers.

[8] Demoen, B., and Sagonas, K. CHAT: The copy-hybrid approach to tabling. In *Proceedings of Practical Aspects of Declarative Programming (PADL)* (1999), LNCS 1551, Springer-Verlag, pp. 106–121.

[9] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. *Biological sequence analysis*. Cambridge University Press, 1998. ISBN: 0–521–62971–3.

[10] Eddy, S. R. Profile hidden markov models (review). *Bioinformatics 14*, 9 (1998), 755–763.

[11] Frost, D., and Dechter, R. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'95* (Aug. 1995), pp. 572–578.

[12] Guo, H.-F., and Gupta, G. A simple scheme for implementing tabled logic programming systems based on dynamic reordering of alternatives. In *Proceedings International Conference on Logic Programming (ICLP)* (2001), LNCS 2237, Springer-Verlag, pp. 181–195.

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

[14] Koller, D. Probabilistic relational models. In *Proceedings of the 9th International Workshop on Inductive Logic Programming* (1999), S. Džeroski and P. Flach, Eds., vol. 1634 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 3–13. Invited paper.

[15] Krogh, A. An introduction to hidden markov models for biological sequences. In *Computational Methods in Molecular Biology* (Amsterdam, 1998), S. L. Salzberg, D. B. Searls, and S. Kasif, Eds., Elsevier, pp. 45–63.

[16] Kumar, V. Algorithms for constraint satisfaction problems: A survey. *AI Magazine 13* (1992), 32–44.

[17] Liu, M. Deductive database languages: Problems and solutions. *ACM Computing Surveys 31*, 1 (1999), 27–62.

[18] Manning, C. D., and Schütze, H. *Foundations of Statistical Natural Language Processing*. The MIT Press, 2000.

[19] Michie, D. "memo" functions and machine learning. *Nature* (1968), 19–22.

[20] Muggleton, S. Stochastic logic programs. In *Advances in Inductive Logic Programming*, L. D. Raedt, Ed. IOS Press, 1996, pp. 254–264.

[21] Park, S. A calculus for probabilistic languages. In *Proceedings of the 2003 ACM SIGPLAN international workshop on Types in languages design and implementation* (2003), ACM Press, pp. 38–49.

[22] Parker, D., Carey, M., Jarke, M., Sciore, E., and Walker, A. Logic programming and databases. In *Expert Database Systems* (1986).

[23] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, 1987.

[24] Pfeffer, A., Koller, D., Milch, B., and Takusagawa, K. T. Spook: A system for probabilistic object-oriented knowledge representation. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)* (1999), K. B. Laskey and H. Prade, Eds., Morgan Kaufmann Publishers, pp. 541–550.

[25] Poole, D. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence 64*, 1 (1993), 81–129.

[26] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recoginition. *Proceedings of the IEEE 77* (1989), 257–286.

[27] Raedt, L. D., and Kersting, K. Probabilistic logic learning. *SIGKDD Explorations 5* (2003), 31–48.

[28] Ramakrishnan, C. Model checking with tabled logic programming. In *ALP News Letter* (2002), ALP.

[29] Ramakrishnan, R., and Ullman, J. D. A survey of deductive database systems. *Journal of Logic Programming 23*, 2 (May 1995), 125–149.

[30] Ramsey, N., and Pfeffer, A. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (2002), ACM Press, pp. 154–165.

[31] Rocha, R., Silva, F., and Costa, V. S. On a tabling engine that can exploit or-parallelism. In *Proceedings International Conference on Logic Programming (ICLP)* (2001), LNCS 2237, Springer-Verlag, pp. 43–58.

[32] Sagonas, K., and Swift, T. An abstract machine for tabled execution of fixed-order stratified logic programs. *ACM Transactions on Programming Languages and Systems 20*, 3 (1998), 586–634.

[33] Sato, T. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming* (1995), L. Sterling, Ed., MIT Press, pp. 715–730.

[34] Sato, T., and Kameya, Y. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* (2001), 391–454.

[35] Shen, Y.-D., Yuan, L., You, J., and Zhou, N.-F. Linear tabulated resolution based on Prolog control strategy. *Theory and Practice of Logic Programming (TPLP) 1*, 1 (2001), 71–103.

[36] Tamaki, H., and Sato, T. OLD resolution with tabulation. In *Proceedings of the Third International Conference on Logic Programming* (1986), E. Shapiro, Ed., LNCS, Springer-Verlag, pp. 84–98.

[37] Thrun, S. Probabilistic robotics. *Communications of the ACM 45*, 3 (Mar. 2002), 52–57.

[38] Tsang, E. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[39] Ullman, J. D. *Database and Knowledge-Base Systems*, vol. 1 & 2. Computer Science Press, 1988.

[40] Uratani, N., Takezawa, T., Matsuo, H., and Morita, C. ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, 1994. In Japanese.

[41] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory IT-13* (Apr. 1967), 260–269.

[42] Warren, D. S. Memoing for logic programs. *Comm. of the ACM, Special Section on Logic Programming 35*, 3 (1992), 93.

[43] Waterman, M. S. *Introduction to Computational Biology: Maps, Sequences and Genomes*. CRC Press, 1995.

[44] Wetherell, C. S. Probabilistic languages: A review and some open questions. *ACM Computing Surveys 12*, 4 (1980), 361–379.

[45] Zhou, N.-F. A logic programming approach to channel routing. In *Proceedings of the 12th International Conference on Logic Programming* (June 13–18 1995), L. Sterling, Ed., MIT Press, pp. 217–232.

[46] Zhou, N.-F. Implementing constraint solvers in B-Prolog. In *IFIP World Congress, Intelligent Information Processing*. Kluwer Academic Publishers, 2002, pp. 249–260.

[47] Zhou, N.-F. CGLIB — a constraint-based graphics library. *Software Practice and Experience 33*, 13 (Nov. 2003), 1199–1216.

[48] Zhou, N.-F., and Sato, T. Efficient fixpoint computation in linear tabling. In *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming* (2003), pp. 275–283.

[49] Zhou, N.-F., Sato, T., and Hasida, K. Toward a high-performance system for symbolic and statistical modeling. In *IJCAI Workshop on Learning Statistical Models from Relational Data* (2003), pp. 153–159.

[50] Zhou, N.-F., Shen, Y.-D., Yuan, L., and You, J. Implementation of a linear tabling mechanism. *Journal of Functional and Logic Programming 2001(1)* (2001), 1–15.