Efficient EM Learning with Tabulation for Parameterized Logic Programs

Yoshitaka KAMEYA and Taisuke SATO

Dept. of Computer Science, Graduate School of Information Science and Engineering,
Tokyo Institute of Technology,
Ookayama 2-12-1, Meguro-ku, Tokyo, Japan, 152-8552.
kame@mi.cs.titech.ac.jp, sato@cs.titech.ac.jp

Abstract. We have been developing a general symbolic-statistical modeling language [6, 19, 20] based on the logic programming framework that semantically unifies (and extends) major symbolic-statistical frameworks such as hidden Markov models (HMMs) [18], probabilistic context-free grammars (PCFGs) [23] and Bayesian networks [16]. The language, PRISM, is intended to model complex symbolic phenomena governed by rules and probabilities based on the *distributional semantics*[19]. Programs contain statistical parameters and they are automatically learned from randomly sampled data by a specially derived EM algorithm, the graphical EM algorithm. It works on support graphs representing the shared structure of explanations for an observed goal. In this paper, we propose the use of tabulation technique to build support graphs, and show that as a result, the graphical EM algorithm attains the same time complexity as specilized EM algorithms for HMMs (the Baum-Welch algorithm [18]) and PCFGs (the Inside-Outside algorithm [1]).

1 Introduction

We have been developing a general symbolic-statistical modeling language [19, 20, 6] based on the logic programming framework that semantically unifies (and extends) major symbolic-statistical frameworks such as hidden Markov models (HMMs) [18], probabilistic context-free grammars (PCFGs) [23] and Bayesian networks [16]. The language, PRISM (***programming in statistical modeling***), is intended to model complex symbolic phenomena governed by rules and probabilities using the *distributional semantics*[19]. Programs contain statistical parameters and they are automatically learned from randomly sampled data by a specially derived EM algorithm, the *graphical EM algorithm*. It works on *support graphs* representing the shared structure of explanations for an observed goal. In this paper, we propose the use of tabulation technique to build support graphs, and show that as a result, the graphical EM algorithm attains the same time complexity as specilized EM algorithms for HMMs (the Baum-Welch algorithm [18]) and PCFGs (the Inside-Outside algorithm[1]). Our subject in this paper is inter-deciplinary, concerning logic programming, probability theory, statistics and formal languages, and the reader is assumed to be familiar with basics of these deciplines [5, 11, 18, 22, 23].

The rest of this paper is as follows. After having a look at background in the next section, and preparing basic materials of PRISM in Sec 3, we present an efficient learning algorithm for PRISM (Sec. 4). We evaluate the time complexity of our algorithm in Sec. 5. Sec. 6 contains a conclusion. Throughout the paper, we use Prolog conventions for logic programs.

2 Background

2.1 Constraint approach and distribution approach

Since our work is at the crossroads of logic programming and probability, it might help to first review various attempts made to integrate probability with logic, or logic programming (though we do not claim exhaustiveness of the list at all). In reviewing, one can immediately notice there are two different basic attitudes towards the use of probability in logic or logic programming. One type, *constraint approach*, emphasizes the role of probabilities as constraints and does not necessarily seek for a unique probability distribution over logical formulas. The other type, *distribution approach*, explicitly defines a unique distribution by model theoretical means or proof theoretical means, to compute various probabilities of propositions.

A typical constraint approach is seen in the early work of probabilistic logic by Nilsson [15]. He considered probabilities assigned to formulas in a knowledge base as constraints on the possible range of probability of a formula of interest. He used linear programming techniques to solve constraints that necessarily delimits the applicability of his approach to finite domains. Turining to logic programming, probabilistic logic programming formalized by Ng and Subrahmanian used clauses of the form $A : \mu \leftarrow F_1 : \mu_1, \ldots, F_n : \mu_n$ annotated by probability intervals μ_is [13]. Lakshmanan and Sadri also used annotated clauses $A \stackrel{c}{\leftarrow} B_1, \ldots, B_n$ where $c = \langle I_b, I_d \rangle$ in the formalization of their probabilistic logic programming. Here I_b represents an expert's belief interval, I_d a doubt interval respectively [10]. Both formalizations only allowed for a finite number of constant and predicate symbols, but no function symbols[10, 13].

Some of the early works of the distribution approach to combining probability with logic programs came out of the Bayesian network community[1]. Breese made a first attempt to use logic programs to automatically build a Bayesian network from a query [2]. After identifying atoms relevant to the query, a local Baysian network for them is constructed to compute posterior probabilities. Logical variables can appear in atoms but no function symbol was allowed [2] (see also [14] for recent development of the use of logic programs to build Bayesian networks). Poole proposed "probabilistic Horn abduction" [17]. His program consists of definite clauses and *disjoint declarations* of the form `disjoint([`h_1`:`p_1`, ... ,`h_n`:`p_n`])` which specifies a probability distribution over the hypotheses (abducibles) $\{h_1, \ldots, h_n\}$. He assigned unique probabilities to all

[1] A Bayesian network is a finite directed acyclic graph representing probabilistic dependences between (continuous or discrete) random variables [16]

ground atoms with the help of the theory of logic programming, and furthermore proved that Bayesian networks are representable in his framework [17]. He however imposed various conditions (the covering property, the acyclicity property, etc [17]) on the class of applicable programs.

In a more linguistic vein, Muggleton formulated SLP (Stochastic Logic Programming) procedurally, as an extension of logic programming to PCFGs [12]. So, a clause C, which must be range-restricted[2], is annotated with a probability p like $p : C$. The probability of a goal G is the products of such ps appearing in its derivation, but with a modification such that if a subgoal g can invoke n clauses, $p_i : C_i$ $(1 \leq i \leq n)$ at some derivation step, the probability of choosing k-th clause is normalized, that is $p_k / \sum_{i=1}^{n} p_i$. SLP was further extended by Cussens by introducing the notion of loglinear models for SLD refutation proofs and defining probabilities of ground atoms in terms of their SLD-trees and "features"[3].

2.2 Limitations and problems

Approaches described so far have more or less similar limitations and problems. Descriptive power confined to finite domains is the most common limitation, which is due to the use of linear programming techniques [15], or due to the syntactic restrictions not allowing for infinitely many constant, function or predicate symbols [10, 13]. Bayesian networks have the same limitation as well (only a finite number of random variables are representable). Also there are various semantic/syntactic restrictions on logic programs. For instance the acyclicity condition [14, 17] prevents the use of clauses with local variables unconditionally, and the range-restrictedness [3, 12] excludes programs such as the usual membership Prolog program. These restrictions would cause problems when we model the distribution of infinitely many objects such as natural language sentences [11].

There is another type of problem, the inconsistent assignment of probabilities. Think of extensions of PCFGs to logic programs [3, 12]. Since they define the probability $\mathbf{Pr}(A)$ of an atom A in terms of syntactic features of the proof trees for A, it is quite possible for $\mathbf{Pr}(A)$ and $\mathbf{Pr}(A \wedge A)$ to differ as their proof trees are different, though logically, they are one and the same.

Last but not least, there is a big problem common to any approach using probabilities: *where do the numbers come from*? Generally speaking, if we use n binary random variables, we need to get 2^n probabilities to completely specify their joint distribution, and this kind of attempt quickly becomes impossible as n grows. Also if there are "hidden variables" in the model such as true causes of a disease, we need lots of work to get reliable probabilities of those variables. Despite these difficulties, all approaches in subsection 2.1 assume *their numbers are given a priori*, and none of them address the problem of how to find probabilities, excepts attempts to use learning techniques for Bayesian networks [8].

[2] A syntactic property that variables appearing in the head also appear in the body of a clause. So, a unit clause must be ground.

2.3 The idea of PRISM

We have been developing a general symbolic-statistical modeling language called PRISM since 1995 along the line of the distribution approach that is free of limitations mentioned above [19, 20]. It is a probabilistic logic programming language equipped with a new semantic framework termed the *distributional semantics* [19], an extension of the least Herbrand model semantics to possible world semantics with a distribution. Theoretically, a PRISM program $DB = F \cup R$ is comprised of a denumerable[3] number of ground facts (hypotheses, abducibles) F and a denumerable number of rules (definite clauses) R in a first-order language with a denumerable number of constant symbols, function symbols and predicate symbols. F is supposed to come with a *basic distribution* P_F that is a completely additive probability measure. So every ground atom in F is considered a random variable taking on 1 (true) or 0 (false). A sampling of P_F determines a set F' of true atoms, which in turn determines the set of true atoms as $M_{DB}(F' \cup R)$ where M_{DB} denotes the least Herbrand model of $F' \cup R$. Hence, every ground atom in DB is a random variable. Their joint distribution P_{DB}, a completely additive probability measure as an extension of P_F, is defined to be the denotation (declarative semantics) of DB (the *distributional semantics*) [19].

Thanks to a general semantic framework, we need none of restrictions such as no function symbols and a finite number of constant and predicate symbols [2, 4, 8, 10, 13], the acyclicity [14, 17] and the covering assumption [17], the range-restrictedness of clauses [3, 12], or the finiteness of domains [2, 4, 15]. A user can write whatever program he/her likes at their own risk without a fear of inconsistent probabilities. Also we succeeded in deriving a new EM learning algorithm for learning statistical parameters in PRISM programs (BS programs) [19], and hence every program can learn from positive examples. So far, we have confirmed the descriptive/learning power of PRISM by tackling various domains including three major symbolic-statistical models, HMMs, PCFGs and Bayesian networks [20].[4]

2.4 Problem of computational complexity

The major problem with the current implementation of PRISM is the slow speed of learning. After determining the scope of symbolic-statistical phenomena we model such as stochastic language generation, we write a parameterized model DB $(= F \cup R)$ that can explain conceivable observations (sentences for instance). If independent observations $G_1, \ldots, G_T$ are given, we let our EM algorithm to learn statistical parameters in DB by (locally) maximizing $\prod_t P_{DB}(G_t = 1)$. The learning process starts with collecting all such $S \subseteq F$ that $S \cup R \models G$ for each observation G_t. This all solution search usually contains a lot of redundancy,

[3] We hereafter use a term "denumerable" as a synonym of "countably infinite." The finite case is similarly treated.

[4] Koller has proposed a probabilistic functional language [9] which can represent HMMs, PCFGs and Bayesian networks, but left the problem of declarative semantics and learning untouched.

and in case of HMMs, we end up in the time complexity exponential in the length of an input string. The reason is obvious: in the stochastic automata such as HMMs, the number of transition paths is exponential, but the Baum-Welch algorithm [18], the specialized EM algorithm for HMMs, achieves linear time complexity by taking advantage of structure-sharing of transition paths represented as a trellis diagram, which corresponds to the reuse of solved subgoals in logic programming. We therefore introduced a reuse mechanism of solved goals such as OLDT search [21] in PRISM, and thus rederived the whole EM algorithm to combine with the OLDT search. Owing to this entire reconstruction, our EM algorithm, though applicable to even type-0 stochastic grammars, has achieved the same time complexity as specialized EM algorithms as far as HMMs and PCFGs are concerned, as described in the sequel.

3 PRISM programs

In this section, we define PRISM programs and the related concepts. See also the basic idea of the distributional semantics in Sec. 2.3.

Definition 1. *A PRISM program is a definite clause program $DB = F \cup R$ which satisfies the following conditions on facts F, their distribution P_F and rules R.*

1. *F is a set of ground atoms of the form* `msw(`i`,`n`,`v`)`*. The arguments i and n are called* group-id *(or* switch name*) and* trial-id*, respectively. We assume that a finite set V_i of ground terms is associated with each i, and $v \in V_i$ holds.*[5] *V_i corresponds to a set of possible values of switch i.*
2. *Let V_i be $\{v_1, v_2, \ldots, v_{|V_i|}\}$. Then, one of the ground atoms* `msw(`i`,`n`,`v_1`)`*,* `msw(`i`,`n`,`v_2`)`*, ...,* `msw(`i`,`n`,`$v_{|V_i|}$`)` *becomes exclusively true (takes the value 1) on each trial. For each i, $\theta_{i,v} \in [0,1]$ is a* parameter *of the (marginal) probability of* `msw(`i`,`$\cdot$`,`v`)` *being true ($v \in V_i$), and $\sum_{v \in V_i} \theta_{i,v} = 1$ holds.*
3. *For each ground terms i, i', n, n', $v \in V_i$ and $v' \in V_{i'}$, random variable* `msw(`i`,`n`,`v`)` *is independent of* `msw(`i'`,`n'`,`v'`)` *if $n \neq n'$ or $i \neq i'$.*
4. *Define $head(R)$ as a set of atoms appearing in the head of R. Then, $F \cap head(R) = \emptyset$.*

In the first condition, we introduce a predicate `msw/3` to represent a basic probabilistic choice such as coin-tossing (`msw` stands for *multi-valued switch*). A ground atom `msw(`i`,`n`,`v`)` represents an event "a switch named i takes on v as a sample value on the trial n." We combine these switches to build a probability distribution of complex phenomena. The second and the third condition say that a logical variable `V` in `msw(`i`,`n`,V)` behaves like a random variable which is realized to v_k with probability θ_{i,v_k} ($k = 1 \ldots |V_i|$).[6] Moreover, from the third condition, the logical variables `V1` and `V2` in `msw(`i`,`n_1`,V1)` and `msw(`i`,`n_2`,V2)` can

[5] As described before, we consider DB as a denumerable set of ground clauses, i and n are arbitrary ground terms in the Herbrand universe.

[6] These probabilities are either learned or given by the user.

be seen as *independent and identically distributed* (i.i.d.) random variables if n_1 and n_2 are different ground terms. The fourth condition says that no $\mathtt{msw}(\cdot,\cdot,\cdot)$ appears in the head of R.

3.1 A program example

We here pick up a PRISM program which represents an HMM, also known as a probabilistic regular grammar. HMMs define a probability distribution over the strings of given alphabets, and can be considered as probabilistic string generators [18], in which an output string is a sample from the defined distribution. The HMM represented below[7] has two states $\{\mathtt{s0},\mathtt{s1}\}$ and outputs a symbol `a` or `b` in each state. For simplicity, the length of output strings is fixed to three.

```
(1) target(hmm/1).               (4) values(init,[s0,s1]).
(2) data('hmm.dat').             (5) values(out(_),[a,b]).
(3) table([hmm/1,hmm/3]).        (6) values(tr(_),[s0,s1]).
(7) hmm(Cs):-                    % To generate a string (chars) Cs...
       msw(init,null,Si),        %     Set initial state to Si, and then
       hmm(1,Si,Cs).             %     Enter the loop with clock = 1.
(8) hmm(T,S,[C|Cs]):- T=<3,      % Loop:
       msw(out(S),T,C),          %     Output C in state S.
       msw(tr(S),T,NextS),       %     Transit from S to NextS.
       T1 is T+1,                %     Put the clock ahead.
       hmm(T1,NextS,Cs).         %     Continue the loop (recursion).
(9) hmm(T,_,[]):- T>3.           % Finish the loop if clock > 3.
```

Procedurally, the above HMM program simulates the generation process of strings (see the comments in the program). Clauses `(7)`∼`(9)` represent the probabilistic behavior of the HMM. In clause (8), to output a symbol `C`, we use different switches `out(S)` conditional on the state `S`.[8] Note that `T` in `msw(out(S),T,C)` is used to guarantee the independency among the choices at each time step. Recursive clauses like `(8)` are allowed in the distributional semantics, and so are in PRISM. Clauses `(1)`∼`(6)` contain additional information about the program. Clause `(1)` declares only the ground atoms containing `hmm`/1 are observable. `hmm([a,b,a])` being true means this HMM generates the string `aba`. Clause `(2)` specifies a file storing learning data. Clause `(3)` specifies the table predicates (described later) are `hmm`/1 and `hmm`/3. We can read that $V_{\mathtt{init}} = \{\mathtt{s0},\mathtt{s1}\}$, $V_{\mathtt{out}(\cdot)} = \{\mathtt{a},\mathtt{b}\}$, $V_{\mathtt{tr}(\cdot)} = \{\mathtt{s0},\mathtt{s1}\}$ from clauses `(4)`∼`(6)`.

3.2 Further definitions and assumptions

For the learning algorithm for PRISM, we need some definitions and assumptions. For the moment, we assume the set I of group-ids coincides with the

[7] The clause numbers are not written in the actual program.

[8] Generally speaking, a conditional probability table (CPT) of a random variable X can be represented by the switch $\mathtt{msw}(f(c_1,c_2,\ldots,c_n),\cdot,x)$, where n is the number of conditional variables, f is the id of X, c_i ($i = 1 \ldots n$) is the value of each conditional variable C_i, and x is one of X's possible values $x_1, x_2, \ldots, x_k$. Of course, $V_{f(c_1,c_2,\ldots,c_n)} = \{x_1, x_2, \ldots, x_k\}$ should be declared in advance.

Herbrand universe of DB. Based on I, a(-n infinite-dimension) *parameter space* Θ is defined as follows:

$$\Theta \stackrel{\text{def}}{=} \prod_{i\in I}\{\langle\theta_{i,v_1},\ldots,\theta_{i,v_{|V_i|}}\rangle \mid V_i=\{v_1,\ldots,v_{|V_i|}\},\ \textstyle\sum_{v\in V_i}\theta_{i,v}=1\}. \quad (1)$$

We next define the probabilistic inconsistency (consistency), probabilistic exclusiveness, and independency w.r.t. facts and goals.

Definition 2. *Consider a PRISM program $DB = F \cup R$ and a set S of facts in F ($S \subseteq F$). S is said to be* p-inconsistent *if $P_F(\boldsymbol{S}=\mathbf{1}|\boldsymbol{\theta}) = 0$ for any parameters $\boldsymbol{\theta} \in \Theta$.*[9] *Otherwise, S is said to be* p-consistent. *Consider two sets S_1 and S_2 of facts in F, which are p-consistent. Then S_1 is said to be* p-exclusive *to S_2 if $S_1 \cup S_2$ is p-inconsistent. Furthermore, let B_1 and B_2 be arbitrary two atoms in $head(R)$. Then, B_1 is said to be* p-exclusive *to B_2 if and only if $P_{DB}(B_1 = 1, B_2=1|\boldsymbol{\theta}) = 0$ for any $\boldsymbol{\theta} \in \Theta$.*

Definition 3. *For each B in $head(R)$, let $S^{(1)},\ldots,S^{(m)}$ be minimal subsets of F such that*

$$comp(R) \models B \leftrightarrow S^{(1)} \vee \cdots \vee S^{(m)}, \quad (2)$$

where $0 \leq m$ and $comp(R)$ is the completion [5] *of R.*[10] *Then, each of $S^{(1)},\ldots,S^{(m)}$ is referred to as a* minimal support set *or an* explanation *for B. We put $\psi_{DB}(B) = \{S^{(1)},\ldots,S^{(m)}\}$.*

Together with a PRISM program $DB = F \cup R$, we always consider a (denumerable) subset $obs(DB)$ of $head(R)$, which is referred to as a set of observable atoms. Each $G \in obs(DB)$ is called a *goal*. Note that the following assumptions are made only for *practical* reasons (e.g. program termination and efficiency), and that the distributional semantics itself does not require these assumptions.

Assumption 1. *Consider a PRISM program DB. In Eq. 2, m is finite ($m < \infty$), and each of $S^{(1)}$, ..., $S^{(m)}$ is a finite set* (finite support condition). *For any $G \in obs(DB)$, explanations in $\psi_{DB}(G)$ are p-consistent and p-exclusive* (exclusiveness condition). *Goals in $obs(DB)$ are p-exclusive to each another, and $\sum_{G\in obs(DB)} P_{DB}(G=1|\boldsymbol{\theta}) = 1$ holds for* some *parameter $\boldsymbol{\theta} \in \Theta$* (uniqueness condition).

From the uniqueness condition, we know that just one atom in $obs(DB)$ becomes true at each observation. Suppose we make T ($< \infty$) independent observations, and G_t is the atom obtained at the t-th observation ($G_t \in obs(DB)$, $t = 1\ldots T$). *Observed data* $\mathcal{G}$ is a finite sequence $\langle G_1, G_2, \ldots, G_T\rangle$. Then, I is redefined here as a set of the group-ids of relevant switches to $\mathcal{G}$, i.e. $I \stackrel{\text{def}}{=} \bigcup_{t=1}^{T}\bigcup_{S\in\psi_{DB}(G_t)}\{i \mid \exists n, v(\mathtt{msw}(i,n,v) \in S)\}$. Also, we redefine Θ as the (finite-dimension) parameter space by Eq. 1.

[9] $\boldsymbol{S}$ is a random vector whose elements are in S. $\mathbf{1}$ (resp. $\mathbf{0}$) is a vector consisting of all 1s (resp. 0s). $\boldsymbol{S}=\mathbf{1}$ means all atoms in S are true.

[10] We sometimes consider a conjunction of atoms $A_1, A_2, \ldots$ as a set $\{A_1, A_2, \ldots\}$.

4 Learning PRISM programs

Learning a PRISM program means *maximum likelihood estimation* (MLE) of the parameters in the program. That is, given observations $\mathcal{G} = \langle G_1, \ldots, G_T \rangle$, we find the parameter $\boldsymbol{\theta} \in \Theta$ which (locally) maximizes the *likelihood* $\Lambda(\mathcal{G}|\boldsymbol{\theta}) \stackrel{\text{def}}{=} \prod_{t=1}^{T} P_{DB}(G_t = 1|\boldsymbol{\theta})$.[11] Although the PRISM program is affected by the behavior, hence by the parameters $\boldsymbol{\theta}$ of switches `msw(·,·,·)` it contains, we cannot directly observe their behavior (i.e. these switches are "hidden"). Hence we apply the EM algorithm [22]. The learning procedure comprises two phases:

- *Find* all explanations $\psi_{DB}(G_t)$ for each goal G_t $(t = 1 \ldots T)$.
- *Run* the EM algorithm based on the statistics from $\psi_{DB}(G_t)$ $(t = 1 \ldots T)$.

In the rest of this section, we first quickly derive a naive version of the EM algorithm,[12] assuming ψ_{DB}. We then introduce *support graphs*, a compact data-structure for ψ_{DB}. After the introduction of support graphs, the *graphical EM algorithm*, an efficient EM algorithm working on support graphs, is described.

4.1 Naive approach

To derive an EM algorithm for PRISM, we must define a *Q function*. First, from the exclusiveness and the uniqueness condition, it is easily shown that explanations in $\Delta_{DB} \stackrel{\text{def}}{=} \bigcup_{G \in obs(DB)} \psi_{DB}(G)$ are all p-exclusive each other. Besides, also from the uniqueness condition,

$$\begin{aligned}\textstyle\sum_{G \in obs(DB)} P_{DB}(G\!=\!1|\boldsymbol{\theta}) &= \textstyle\sum_{G \in obs(DB)} \sum_{S \in \psi_{DB}(G)} P_{DB}(\boldsymbol{S}\!=\!\mathbf{1}|\boldsymbol{\theta}) \\ &= \textstyle\sum_{S \in \Delta_{DB}} P_{DB}(\boldsymbol{S}\!=\!\mathbf{1}|\boldsymbol{\theta}) = 1\end{aligned}$$

holds for any $\boldsymbol{\theta} \in \Theta$. Hence, exactly one of the explanations in Δ_{DB} is true. Since Δ_{DB} is denumerable, we can consider an isomorphic map $f : \Delta_{DB} \to \boldsymbol{N}^{+}$, where $\boldsymbol{N}^{+}$ is a set of positive integers, and temporarily introduce a new random variable E on Ω_F such that $E = f(S)$ if $S \in \Delta_{DB}$ is exclusively true ($\boldsymbol{S} = \mathbf{1}$), or $E = 0$ otherwise. Now we are in a position to define the Q function:

$$Q(\boldsymbol{\theta}', \boldsymbol{\theta}) \stackrel{\text{def}}{=} \textstyle\sum_{t=1}^{T} \sum_{e \in \boldsymbol{N}} P_{DB}(E\!=\!e|G_t\!=\!1, \boldsymbol{\theta}) \log P_{DB}(E\!=\!e, G_t\!=\!1|\boldsymbol{\theta}'), \quad (3)$$

where $\boldsymbol{N}$ is a set of non-negative integers. It is easy to show $Q(\boldsymbol{\theta}', \boldsymbol{\theta}) \geq Q(\boldsymbol{\theta}, \boldsymbol{\theta}) \Rightarrow P_{DB}(G_t = 1|\boldsymbol{\theta}') \geq P_{DB}(G_t = 1|\boldsymbol{\theta})$. Therefore, for MLE, starting with some parameters $\boldsymbol{\theta}^{(0)}$, we iteratively update parameters by $\boldsymbol{\theta}^{(m+1)} := \mathrm{argmax}_{\boldsymbol{\theta}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(m)})$

[11] Under the exclusiveness condition, each marginal probability of G_t being true is calculated as below. $\sigma_{i,v}(S)$ is defined as $\sigma_{i,v}(S) \stackrel{\text{def}}{=} \left|\left\{n \mid \texttt{msw}(i,n,v) \in S\right\}\right|$.

$$P_{DB}(G_t\!=\!1|\boldsymbol{\theta}) = \textstyle\sum_{S \in \psi_{DB}(G_t)} P_F(\boldsymbol{S}\!=\!\mathbf{1}|\boldsymbol{\theta}) = \sum_{S \in \psi_{DB}(G_t)} \prod_{i \in I, v \in V_i} \theta_{i,v}^{\sigma_{i,v}(S)},$$

[12] In [6], PRISM* programs are introduced to remove computationally intractable terms. We here present an alternative way to remove them.

until the *log-likelihood* $\log \Lambda(\mathcal{G}|\boldsymbol{\theta})$ converges. Transforming Eq. 3, the following formula is obtained:

$$Q(\boldsymbol{\theta}', \boldsymbol{\theta}) = \sum_{i\in I, v\in V_i} \eta(i, v, \boldsymbol{\theta}) \log \theta'_{i,v} \leq \sum_{i,v} \left(\eta(i, v, \boldsymbol{\theta}) \log \frac{\eta(i,v,\boldsymbol{\theta})}{\sum_{v'\in V_i} \eta(i,v',\boldsymbol{\theta})} \right),$$

where $\eta(i, v, \boldsymbol{\theta}) \stackrel{\text{def}}{=} \sum_{t=1}^{T} \frac{1}{P_{DB}(G_t=1|\boldsymbol{\theta})} \sum_{S\in\psi_{DB}(G_t)} P_F(\boldsymbol{S}=\mathbf{1}|\boldsymbol{\theta})\sigma_{i,v}(S)$. Hence, we reach the procedure *learn-naive* in Fig. 1 that finds the MLE of the parameters. The array variable $\eta[i, v]$ contains $\eta(i, v, \boldsymbol{\theta})$ under the current $\boldsymbol{\theta}$. In this procedure, the calculations for $P_{DB}(G_t = 1|\boldsymbol{\theta})$ and $\eta[i, v]$ (Line 2, 5 and 9) are computationally intractable when $|\psi_{DB}(G_t)|$ is exponential (though finite) in the complexity of the model.[13]

4.2 Tabulation approach

For efficient computation of $P_{DB}(G_t = 1|\boldsymbol{\theta})$ and $\eta[i, v]$, we introduce structure-sharing of explanations by tabulation, which requires more assumptions on DB. We assume that a set of table predicates $\mathit{table}(DB)$ is declared in advance (like the HMM program in Sec. 3.1). Let τ^*_{DB} be a set of ground atoms containing the table predicate in $\mathit{table}(DB)$. We use $\mathit{comp}(R)$, the completion of rules R, again in the following assumption.

Assumption 2. *Let DB be a PRISM program which satisfies the finite support condition, the exclusiveness condition, and the uniqueness condition. Assume that, for each $t = 1 \ldots T$, the following condition holds for some finite* ordered *subset $\tau^t_{DB} = \{\tau^t_1, \ldots, \tau^t_{K_t}\}$ of τ^*_{DB}:*[14]

$$\begin{aligned} \mathit{comp}(R) \models \Big(G_t \leftrightarrow \tilde{S}^t_{0,1} \vee \cdots \vee \tilde{S}^t_{0,m_0}\Big) & \\ \wedge \Big(\tau^t_1 \leftrightarrow \tilde{S}^t_{1,1} \vee \cdots \vee \tilde{S}^t_{1,m_1}\Big) \wedge \cdots \wedge \Big(\tau^t_{K_t} \leftrightarrow \tilde{S}^t_{K_t,1} \vee \cdots \vee \tilde{S}^t_{K_t,m_{K_t}}\Big) &, \end{aligned} \tag{4}$$

where

- *Letting G_t be τ^t_0, each of $\tilde{S}^t_{k,1}, \ldots, \tilde{S}^t_{k,m_k}$ is a subset of $F \cup \{\tau^t_{k+1}, \ldots, \tau^t_{K_t}\}$, and is also called a* t-explanation[15] *for τ^t_k (for $k = 0 \ldots K_t$). We here put $\tilde{\psi}_{DB}(\tau^t_k) \stackrel{\text{def}}{=} \{\tilde{S}_{k,1}, \ldots, \tilde{S}_{k_{m_k}}\}$ (for $k = 0 \ldots K_t$).*
- *Each of $\tilde{S}^t_{k,1}, \ldots, \tilde{S}^t_{k,m_k}$ $(k = 0 \ldots K_t)$ is a set of independent atoms.*[16]

Each τ^t_k $(k = 1 \ldots K_t)$ is referred to as a table atom*. We call the former condition* acyclic support condition*, and the latter* independent support condition*.*

---

[13] For example, the complexity of the HMM depends on the number of states, the length of input/output string or the number of output alphabets.

[14] From the finite support condition, for $k = 0 \ldots K_t$, m_k is finite and each of $\tilde{S}^t_{k,1}, \ldots, \tilde{S}^t_{k,m_k}$ is finite. Also, from the exclusive condition, $\tilde{S}^t_{k,1}, \ldots, \tilde{S}^t_{k,m_k}$ are p-consistent and p-exclusive $(k = 0 \ldots K_t)$. Besides, from the uniqueness condition, $G_{t'} \notin \tau^t_{DB}$ holds for any $t, t' = 1 \ldots T$.

[15] Prefix "t-" is an abbreviation of "tabled-".

[16] For $B_1, B_2 \in \mathit{head}(R)$, B_1 is independent of B_2 if $P_{DB}(B_1 = y_1, B_2 = y_2|\boldsymbol{\theta}) = P_{DB}(B_1 = y_1|\boldsymbol{\theta}) \cdot P_{DB}(B_2 = y_2|\boldsymbol{\theta})$ for any $y_1, y_2 \in \{0, 1\}$ and any $\boldsymbol{\theta} \in \Theta$.

procedure *learn-naive* $(DB, \mathcal{G})$ **begin**
  Select some $\boldsymbol{\theta}$ from Θ; $\lambda^{(0)} := \sum_{t=1}^{T} \log P_{DB}(G_t = 1|\boldsymbol{\theta})$;
  repeat
    foreach $i \in I, v \in V_i$ **do**
      $\eta[i,v] := \sum_{t=1}^{T} \frac{1}{P_{DB}(G_t=1|\boldsymbol{\theta})} \sum_{S \in \psi_{DB}(G_t)} P_F(\boldsymbol{S}=\boldsymbol{1}|\boldsymbol{\theta})\sigma_{i,v}(S)$;
    foreach $i \in I, v \in V_i$ **do**
      $\theta_{i,v} := \eta[i,v] / \sum_{v' \in V_i} \eta[i,v']$;
    $m := m+1$;
    $\lambda^{(m)} := \sum_{t=1}^{T} \log P_{DB}(G_t = 1|\boldsymbol{\theta})$
  until $\lambda^{(m)} - \lambda^{(m-1)} < \varepsilon$
end.

Fig. 1. A procedure for naive approach.

The task here is to construct such $\tilde{\psi}_{DB}$ and τ_{DB}^{t} from the source PRISM program. One way is to use OLDT (*OLD with tabulation*) [21], a complete search technique for logic programs. In OLDT, a (sub-)goal g containing a table predicate is registered into a *solution table*, whereas the instance of g is registered in a *lookup table*. The latter reuses solutions in the solution table. In what follows, we illustrate our tabulation approach by using the HMM program in Sec. 3.1.

First, we translate the PRISM program to another logic program. Similarly to the translation of *definite clause grammars* (DCGs) in Prolog, we add two arguments (which forms *D-list*) to each predicate for collecting t-explanations. In the case of the HMM program, the translation results in:

```
(T1)  top_hmm(Cs,X):- tab_hmm(Cs,X,[]).
(T3)  tab_hmm(Cs,[hmm(Cs)|X],X):- hmm(Cs,_,[]).
(T3') tab_hmm(T,S,Cs,[hmm(T,S,Cs)|X],X):- hmm(T,S,Cs,_,[]).
(T4)  e_msw(init,T,s0,[msw(init,T,s0)|X],X).
(T4') e_msw(init,T,s1,[msw(init,T,s1)|X],X).
  :
(T7)  hmm(Cs,X0,X1):- e_msw(init,null,Si,X0,X2), tab_hmm(1,Si,Cs,X2,X1).
(T8)  hmm(T,S,[C|Cs],X0,X1):-
          T=<3, e_msw(out(S),T,C,X0,X2), e_msw(tr(S),T,NextS,X2,X3),
          T1 is T+1, tab_hmm(T1,NextS,Cs,X3,X1).
(T9)  hmm(T,S,[],X,X):- T>3.
```

Clauses (Tj) and (Tj') correspond to the original clause (j), respectively. In the translated program, $p/(n+2)$ is a table predicate if p/n is a table predicate in the original program. We use the predicate tab_$p/(n+2)$ to keep the t-explanations (in Eq. 4). Note that tab_$p/(n+2)$ is called instead of the table predicate $p/(n+2)$. We then apply OLDT search while noting (i) added D-list does not influence the original OLDT procedure, and (ii) we associate a list of t-explanations with each solution. For example, running OLDT for the above translated program gives the solution table in Fig. 2. Finally, we extract $\tilde{\psi}_{DB}$, the set of all t-explanations, from this table. The remaining task is to get totally ordered table atoms, i.e. the ordered set τ_{DB}^{t}, respecting the acyclicity in Eq. 4. Obviously, it can be done by topological sorting.

```
hmm([a,b,a]): [hmm([a,b,a]):[[m(init,null,s0),hmm(1,s0,[a,b,a])],
                             [m(init,null,s1),hmm(1,s1,[a,b,a])]]]
hmm(1,s0,[a,b,a]):
 [hmm(1,s0,[a,b,a]):[[m(out(s0),1,a),m(tr(s0),1,s0),hmm(2,s0,[b,a])],
                     [m(out(s0),1,a),m(tr(s0),1,s1),hmm(2,s1,[b,a])]]]
hmm(1,s1,[a,b,a]):
 [hmm(1,s1,[a,b,a]):[[m(out(s1),1,a),m(tr(s1),1,s0),hmm(2,s0,[b,a])],
:                    [m(out(s1),1,a),m(tr(s1),1,s1),hmm(2,s1,[b,a])]]]
```

Fig. 2. Solution table (`m` is an abbreviation of `msw`).

To help visualizing our learning algorithm, we introduce a data-structure called *support graphs*, though the algorithm itself is defined using only $\tilde{\psi}_{DB}$ and the ordered set τ^t_{DB}. As illustrated in Fig 3 (a), the support graph for G_t $(t = 1 \ldots T)$, a graphical representation of Eq. 4, consists of disconnected subgraphs, each of which is labeled with the corresponding table atom τ^t_k in τ^t_{DB} $(k = 1 \ldots K_t)$. Each subgraph labeled τ^t_k comprises two special nodes, the *start node* and the *end node*, and *explanation graphs*, each of which corresponds to a t-explanation $\tilde{S}^t_{k,j}$ in $\tilde{\psi}_{DB}(\tau^t_k)$ $(j = 1 \ldots m_k)$. An explanation graph of $\tilde{S}^t_{k,j}$ is cascaded nodes, where each node is labeled with a table atom τ or a switch $\mathtt{msw}(\cdot,\cdot,\cdot)$ in $\tilde{S}^t_{k,j}$. It is called a *table node* or a *switch node*. Support graphs have a similar structure to *recursive transition networks* (RTNs). Fig 3 (b) is the support graph for `hmm([a,b,a])` obtained from the solution table in Fig 2. Each table node labeled τ refers to the subgraph labeled τ, so data-sharing is achieved by the distinct table nodes referring to the same subgraph.

4.3 Graphical EM algorithm

We describe here a new learning algorithm, the *graphical EM algorithm*, that works on support graphs (more specifically, on $\tilde{\psi}_{DB}$ and τ^t_{DB}). We prepare four arrays for each support graph for G_t $(t = 1 \ldots T)$: $\mathcal{P}[t,\tau]$ for *inside probabilities* of τ, $\mathcal{Q}[t,\tau]$ for *outside probabilities* of τ, $\mathcal{R}[t,\tau,\tilde{S}]$ for *explanation probabilities* of $\tilde{S}$ in $\tilde{\psi}_{DB}(\tau)$, and $\eta[t,i,v]$ for *expected counts* of $\mathtt{msw}(i,\cdot,v)$. The algorithm is

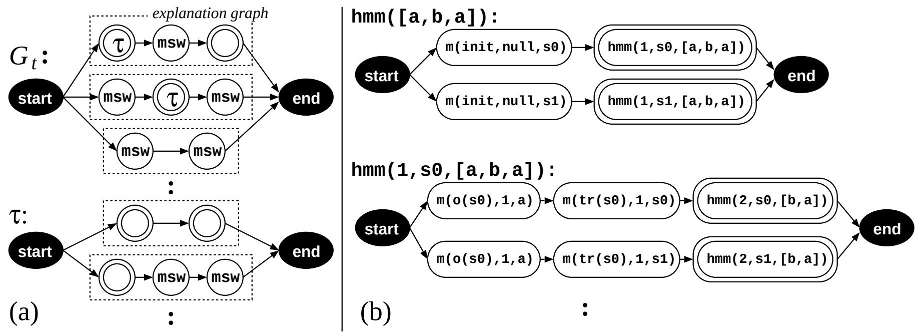

Fig. 3. A support graph (a) in general form, (b) for the HMM program with G_t = `hmm([a,b,a])`. A double-circled node refers to a table node.

1: **procedure** *learn-gEM* $(DB, \mathcal{G})$
2: **begin**
3: Select some $\boldsymbol{\theta}$ from Θ;
4: *get-inside-probs*$(DB, \mathcal{G})$;
5: $\lambda^{(0)} := \sum_{t=1}^{T} \log \mathcal{P}[t, G_t]$;
6: **repeat**
7: *get-expectations*$(DB, \mathcal{G})$;
8: **foreach** $i \in I, v \in V_i$ **do**
9: $\eta[i, v] :=$
10: $\sum_{t=1}^{T} \eta[t, i, v] / \mathcal{P}[t, G_t]$;
11: **foreach** $i \in I, v \in V_i$ **do**
12: $\theta_{i,v} := \eta[i, v] / \sum_{v' \in V_i} \eta[i, v']$;
13: *get-inside-probs*$(DB, \mathcal{G})$;
14: $\lambda^{(m)} := \sum_{t=1}^{T} \log \mathcal{P}[t, G_t]$
15: **until** $\lambda^{(m)} - \lambda^{(m-1)} < \varepsilon$
16: **end.**

procedure *get-inside-probs* $(DB, \mathcal{G})$
begin
for $t := 1$ **to** T **do begin**
Put $G_t = \tau_0^t$;
for $k := K_t$ **downto** 0 **do begin**
$\mathcal{P}[t, \tau_k^t] := 0$;
foreach $\tilde{S} \in \tilde{\psi}_{DB}(\tau_k^t)$ **do begin**
Put $\tilde{S} = \{A_1, A_2, \ldots, A_{|\tilde{S}|}\}$;
$\mathcal{R}[t, \tau_k^t, \tilde{S}] := 1$;
for $\ell := 1$ **to** $|\tilde{S}|$ **do**
if $A_\ell = \mathtt{msw}(i, \cdot, v)$ **then**
$\mathcal{R}[t, \tau_k^t, \tilde{S}] \mathrel{*}= \theta_{i,v}$
else $\mathcal{R}[t, \tau_k^t, \tilde{S}] \mathrel{*}= \mathcal{P}[t, A_\ell]$;
$\mathcal{P}[t, \tau_k^t] \mathrel{+}= \mathcal{R}[t, \tau_k^t, \tilde{S}]$
end /* foreach $\tilde{S}$ ***/**
end /* for k ***/**
end /* for t ***/**
end.

procedure *get-expectations* $(DB, \mathcal{G})$ **begin**
for $t := 1$ **to** T **do begin**
Put $G_t = \tau_0^t$; $\mathcal{Q}[t, \tau_0^t] := 1$; **for** $k := 1$ **to** K_t **do** $\mathcal{Q}[t, \tau_k^t] := 0$;
for $k := 0$ **to** K_t **do**
foreach $\tilde{S} \in \tilde{\psi}_{DB}(\tau_k^t)$ **do begin**
Put $\tilde{S} = \{A_1, A_2, \ldots, A_{|\tilde{S}|}\}$;
for $\ell := 1$ **to** $|\tilde{S}|$ **do**
if $A_\ell = \mathtt{msw}(i, \cdot, v)$ **then** $\eta[t, i, v] \mathrel{+}= \mathcal{Q}[t, \tau_k^t] \cdot \mathcal{R}[t, \tau_k^t, \tilde{S}]$
else $\mathcal{Q}[t, A_\ell] \mathrel{+}= \mathcal{Q}[t, \tau_k^t] \cdot \mathcal{R}[t, \tau_k^t, \tilde{S}] / \mathcal{P}[t, A_\ell]$
end /* foreach $\tilde{S}$ ***/**
end /* for t ***/**
end.

Fig. 4. Graphical EM algorithm.

shown in Fig. 4. Due to the space limitation, details are omitted. It can be shown however that *learn-gEM* is equivalent to the procedure *learn-naive* (Sec. 4.1).[17] As shown in Sec. 4.1, *learn-naive* is the MLE procedure, hence the following theorem holds.

Theorem 1. *Let DB be a PRISM program, and* $\mathcal{G}$ *be the observed data. Then learn-gEM finds* $\boldsymbol{\theta}^* \in \Theta$ *which (locally) maximizes the likelihood* $\Lambda(\mathcal{G}|\boldsymbol{\theta})$.

[17] To be more specific, under the same parameters $\boldsymbol{\theta}$, the value of $\eta[i, v]$ in *learn-naive* (Line 5) is equal to that in *learn-gEM* (Line 10). Hence, the parameters are updated to the same value. Furthermore, starting with the same initial parameters, the converged parameters are also the same.

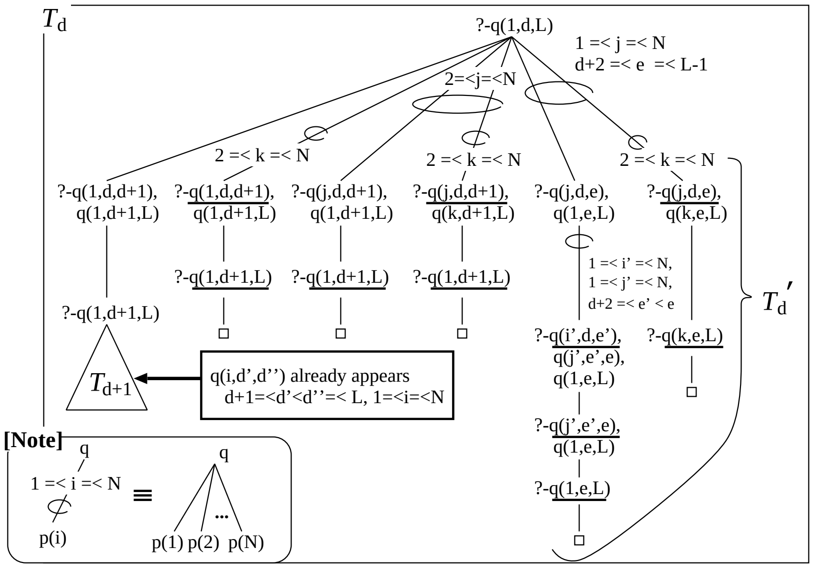

Fig. 5. an OLDT tree T_d for the query `?-q(1,`d`,`L`)`.

5 Complexity

In this section, we estimate the time complexity of our learning method in case of PRISM programs for PCFGs, and compare with the Inside-Outside algorithm. Since our method comprises two phases (OLDT and the graphical EM), we estimate the computation time in each phase.

In the Inside-Outside algorithm, time complexity is measured by N, the number of non-terminals, and L, the number of terminals in the input/output sentence. Assuming that the target grammar is in *Chomsky normal form*, the worst-case time complexity is the computation time for the largest grammar, i.e. a set of all combinations of terminals and non-terminals. Hence, we may start with a logic program (not a PRISMprogram) representing the largest grammar:

$$\begin{aligned} &\left\{ \mathtt{q}(i,d,d')\mathtt{:-}\ \mathtt{q}(j,d,d''),\mathtt{q}(k,d'',d') \;\middle|\; i,j,k = 1\dots N,\ 0 \le d < d'' < d' \le L \right\} \\ &\qquad \cup \left\{ \mathtt{q}(i,d,d') \;\middle|\; i = 1\dots N,\ 0 \le d \le L-1,\ d' = d+1 \right\}. \end{aligned} \tag{5}$$

$\mathtt{q}(i,d,d')$ says that the i-th non-terminal spans from $(d+1)$-th word to d'-th word. The textual order over the clauses "$\mathtt{q}(i,d,d')\mathtt{:-}\ \mathtt{q}(j,d,d''),\mathtt{q}(k,d'',d')$" is the lexicographic order over the tuples (i,j,k,d,d',d''). We then make an exhaustive search for the query by OLDT. Assuming that the solution table is accessible in $O(1)$ time, the time complexity of OLDT is measured by the number of nodes in OLDT tree (the search tree for OLDT). We fix the search strategy to *multi-stage depth-first strategy* [21]. Let T_d be an OLDT tree for the query `?-q(1,`d`,`L`)`. Fig. 5 illustrates the case of $0 \le d \le L-3$. As can be seen, even for this simple grammar, the tree has many similar subtrees, so we put them together (see **[Note]** in Fig. 5). Then, due to the depth-first strategy,

T_d has a recursive structure, i.e. T_{d+1} is a part of T_d. We enumerate h_d, the number of the nodes in T_d but not in T_{d+1}. The node with an underlined leftmost atom is a lookup node, which only *consumes* the solution obtained in other place. From Fig. 5, $h_d = O(N^3(L-d)^2)$.[18] Total time for OLDT search is the number of nodes in the OLDT tree for `?-q(1,0,`L`)` (the whole sentence), that is, $\sum_{d=0}^{L-3} h_d = O(N^3L^3)$.[19] In the case of a DCG program below, it can be proved similarly that the time complexity is $O(N^3L^3)$.

$$\begin{aligned}&\left\{\texttt{q(}i\texttt{,L0,L1):- q(}j\texttt{,L0,L2),q(}k\texttt{,L2,L1)} \;\middle|\; i,j,k = 1\ldots N\right\}\\ &\cup \left\{\texttt{q(}i\texttt{,L0,L1):- L0=[}w\texttt{|L1]} \;\middle|\; i = 1\ldots N,\ w \text{ is a terminal symbol}\right\}\end{aligned}$$

Since our method respects the original OLDT procedure, the search time for the corresponding PRISM program given T observed goals is $O(N^3L^3T)$.

On the other hand, the learning time of the graphical EM algorithm is propotional to the size of the support graphs used, i.e. the number of nodes in the graphs. It is easily shown, from the description of *learn-gEM*, that the size of the graphs is $O(\xi_{\mathrm{num}}\xi_{\mathrm{maxsize}}T)$, where $\xi_{\mathrm{num}} \stackrel{\mathrm{def}}{=} \max_{1\le t\le T} |\tilde{\Delta}^t_{DB}|$, $\xi_{\mathrm{maxsize}} \stackrel{\mathrm{def}}{=} \max_{1\le t\le T, \tilde{S}\in\tilde{\Delta}^t_{DB}} |\tilde{S}|$, and $\tilde{\Delta}^t_{DB} \stackrel{\mathrm{def}}{=} \bigcup_{\tau\in\tau^t_{DB}} \tilde{\psi}_{DB}(\tau)$. In the case of the PCFG program, $\xi_{\mathrm{num}} = O(N^3L^3)$ and $\xi_{\mathrm{maxsize}} = O(1)$. Hence, the computation time per update of the graphical EM algorithm is $O(N^3L^3T)$. We therefore have:

Proposition 1. *Let DB be a PRISM program representing a PCFG, and* $\mathcal{G} = \langle G_1, G_2, \ldots, G_T\rangle$ *be the observed data. We assume each table operation in OLDT search is done in time* $O(1)$. *Then OLDT search for the goal* G_t *and each iteration in learn-gEM is done in time* $O(N^3L^3T)$.

$O(N^3L^3T)$ is also the time complexity of the Inside-Outside algorithm, hence our algorithm is as efficient as the Inside-Outside algorithm.[20] Similarly, we can show that, for HMM programs like ones in Sec. 3.1, the search and the learning time is $O(\hat{N}^2LT)$, the same order as that of the Baum-Welch algorithm ($\hat{N}$ is the number of states).

6 Conclusion

We have proposed an efficient EM learning algorithm for the parameterized logic programs which seamlessly unifies logical semantics and probabilistic semantics. It is shown our general algorithm works as efficiently as the specialized EM algorithms such as the Baum-Welch algorithm and the Inside-Outside algorithm.

[18] We here focus on the subtree T'_d. Each of j, i', j' ranges from 1 to N, and $\left|\{(e,e') \mid d+2 \le e' < e \le L-1\}\right| = O((L-d)^2)$. Hence, the number of nodes in T'_d is $O(N^3(L-d)^2)$. The number of nodes in T_d but neither in T_{d+1} nor in T'_d is negligible, therefore $h_d = O(N^3(L-d)^2)$.

[19] The number of nodes of T_{L-1} and T_{L-2} is negligible.

[20] Ours can be better than the Inside-Outside algorithm as only relevant grammar rules are involved in the EM learning phase as is observed experimentally.

Furthermore, due to the generality of the language and the learning algorithm, our framework can be applied to the stochastic grammars with context dependencies such as the bigram model of production rules [7], in which case the time complexity of the EM learning is polynomial (details are omitted). The omitted details in this paper will be included in the full paper we are preparing.

References

1. Baker, J. K., Trainable grammars for speech recognition, *Proc. of Spring Conf. of the Acoustical Society of America*, pp.547–550, 1979.
2. Breese, J. S., Construction of belief and decision networks, *Computational Intelligence*, Vol.8, No.4, pp.624–647, 1992.
3. Cussens, J., Loglinear models for first-order probabilistic reasoning, *Proc. of UAI'99*, pp.126–133, 1999.
4. Dekhtyar, A. and Subrahmanian, V. S., Hybrid probabilistic programs, *Proc. of ICLP'97*, pp.391–405, 1997.
5. Doets, K., *From Logic to Logic Programming*, The MIT Press, 1994.
6. Kameya, Y., Ueda, N. and Sato, T., A graphical method for parameter learning of symbolic-statistical models, *Proc. of DS'99*, LNAI 1721, pp.264–276, 1999.
7. Kita, K., Morimoto, T., Ohkura, K., Sagayama, S. and Yano, Y., Spoken sentence recognition based on HMM-LR with hybrid language modeling, *IEICE Trans. on Information & Systems*, Vol.E77-D, No.2, 1994.
8. Koller, D. and Pfeffer, A., Learning probabilities for noisy first-order rules, *Proc. of IJCAI'97*, pp.1316–1321, 1997.
9. Koller, D., McAllester, D. and Pfeffer, A., Effective Bayesian Inference for Stochastic Programs, *Proc. of AAAI'97*, pp.740–747, 1997.
10. Lakshmanan, L. V. S. and Sadri, F., Probabilistic deductive databases, *Proc. of ILPS'94*, pp.254–268, 1994.
11. Manning, C. D. and Schütze, H., *Foundations of Statistical Natural Language Processing*, The MIT Press, 1999.
12. Muggleton, S., Stochastic logic programs, In *Advances in Inductive Logic Programming* (Raedt,L.De ed.), OSP Press, pp.254–264, 1996.
13. Ng, R. and Subrahmanian, V. S., Probabilistic logic programming, *Information and Computation*, Vol.101, pp.150–201, 1992.
14. Ngo, L. and Haddawy, P., Answering queries from context-sensitive probabilistic knowledge bases, *Theoretical Computer Science*, Vol.171, pp.147–177, 1997.
15. Nilsson, N. J., Probabilistic logic, *Artificial Intelligence*, Vol.28, pp.71–87, 1986.
16. Pearl, J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.
17. Poole, D., Probabilistic Horn abduction and Bayesian networks, *Artificial Intelligence*, Vol.64, pp.81–129, 1993.
18. Rabiner, L. and Juang, B., *Foundations of Speech Recognition*, Prentice-Hall, 1993.
19. Sato, T., A statistical learning method for logic programs with distribution semantics, *Proc. of ICLP'95*, pp.715-729, 1995.
20. Sato, T. and Kameya, Y., PRISM: a language for symbolic-statistical modeling, *Proc. of IJCAI'97*, pp.1330–1335, 1997.
21. Tamaki, H. and Sato, T., OLD resolution with tabulation, *Proc. of ICLP'86*, LNCS 225, pp.84–98, 1986.
22. Tanner, M., *Tools for Statistical Inference* (2nd ed.), Springer-Verlag, 1986.
23. Wetherell, C.S., Probabilistic languages: a review and some open questions, *Computing Surveys*, Vol.12, No.4, pp.361–379, 1980.