A Graphical Method for Parameter Learning of Symbolic-Statistical Models

Yoshitaka KAMEYA, Nobuhisa UEDA, and Taisuke SATO

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

Abstract. We present an efficient method for statistical parameter learning of a certain class of symbolic-statistical models (called PRISM programs) including hidden Markov models (HMMs). To learn the parameters, we adopt the EM algorithm, an iterative method for maximum likelihood estimation. For the efficient parameter learning, we first introduce a specialized data structure for explanations for each observation, and then apply a graph-based EM algorithm. The algorithm can be seen as a generalization of Baum-Welch algorithm, an EM algorithm specialized for HMMs. We show that, given appropriate data structure, Baum-Welch algorithm can be simulated by our graph-based EM algorithm.

1 Introduction

To capture uncertain phenomena in a symbolic framework, we have been developing a symbolic-statistical modeling language PRISM in the past years [10, 11]. PRISM programs is a probabilistic extension of logic programs based on *distributional semantics*, and its programming system has a built-in mechanism for statistical parameter learning from observed data. For parameter learning, we adopt the EM algorithm, an iterative method for maximum likelihood estimation (MLE). With this learning ability built into the expressive power of first-order logic, as shown in [10], PRISM not only covers existing symbolic-statistical models ranging from hidden Markov models (HMMs) [1, 8] to Bayesian networks (BNs) [6] and to probabilistic context-free grammars (PCFGs) [1], but can smoothly model the complicated interaction between gene-inheritance and a tribal social system discovered in the Kariera tribe [11].

Our problem with the current learning algorithm for PRISM is that although our learning is completely general, it lacks the efficiency achieved by other specialized EM algorithms such as the Baum-Welch algorithm [1, 8] for HMMs. We therefore propose an efficient learning framework, in which the learning can be done as efficiently as these specialized EM algorithms without hurting the integration of learning and computing. By inspecting closely our EM algorithm, we have found it is possible to eliminate computationally intractable part of it by imposing a couple of reasonable conditions on modeling, which results in a general but efficient learning algorithm running on a special type of graph structures.

The purpose of this paper is to show mathematically how this improvement becomes possible for what reason. The rest of the paper is organized as follows.

We first modify the semantic framework, slightly, of PRISM in such a way that the modification justifies the elimination of computational redundancy which was inherent in our EM algorithm. Next we introduce a special class of directed acyclic graphs, each of which is a structural representation of explanations for an observation. We call them *support graphs*. We then show that the graph-based EM algorithm, a new EM algorithm implemented on support graphs, attains the same time complexity as the Baum-Welch algorithm.

2 Modifying PRISM to PRISM*

2.1 Distributional semantics

We here shortly describe distributional semantics, the theoretical basis of PRISM (See [9] for details), and some preliminary definitions. A program DB we deal with is written as $DB = F \cup R$ where F is a set of facts (unit clauses) and R is a set of rules (clauses with a non-empty body). In theoretical context, DB is considered as a set of (possibly infinitely many) ground clauses. We define a joint distribution P_F on the set of all possible interpretations for F, and think of each ground atom as a random variable taking 1 when true and 0 otherwise. We call P_F a *basic distribution*. Then, there exists a way to extend P_F to a joint distribution P_{DB} on the set of all possible interpretations for ground atoms appearing in DB. *The denotation of a logic program DB is defined as P_{DB}.*

We here put $head(R)$ as the set of heads appearing in R and then assume that $F \cap head(R) = \emptyset$ holds. In such a case, DB is said to be *separated*. To make matters simple, we further assume that there is a fixed set of ground atoms $G^{patt} \subseteq head(R)$ representing possible observations, and our observation corresponds to randomly picking up one of atoms in G^{patt} as a goal (to be proved by our modeling program). Collecting T observations means statistically making T independent selection of goals G_t ($G_t \in G^{patt}, 1 \leq t \leq T$).

For each $G \in G^{patt}$, we assume there are finite sets $S^{(1)}, \ldots, S^{(m)}$ of ground atoms from F such that $\mathrm{iff}(R) \vdash G \leftrightarrow S^{(1)} \vee \cdots \vee S^{(m)}$, where $\mathrm{iff}(R)$ is the *completion* [2] of R. Each of $S^{(1)}, \ldots, S^{(m)}$ is called a *support set* for G. A *minimal support set* (or an *explanation*) is a support set which is minimal w.r.t. set inclusion ordering. For later use, we introduce $\psi_{DB}(G)$ as a set of minimal support sets of G for $G \in G^{patt}$.

2.2 PRISM programs

PRISM programs must satisfy the following conditions on facts F and the basic distribution P_F.

1. A ground atom in F is of the form `msw(`i`,`n`,`v`)`. It is supposed to represent the fact that a multi-valued probabilistic switch named i yields a value v at n-th sampling. v is taken from V_i, a finite set of ground terms specified beforehand.
2. Put $V_i = \{v_1, v_2, \ldots, v_K\}$. Then exactly one of `msw(`i`,`n`,`v_1`)`, `msw(`i`,`n`,`v_2`)`, $\ldots$, `msw(`i`,`n`,`v_K`)` always holds true. Put differently, $\sum_{v \in V_i} \theta_i(v) = 1$ always holds where $\theta_i(v)$ is the probability of `msw(`i`,`n`,`v`)` ($v \in V_i$) being true. We call $\theta_i(v)$ a *statistical parameter* (or simply a *parameter*) of the program.

3. For $n \neq n'$, $\texttt{msw}(i,n,\cdot)$ and $\texttt{msw}(i,n',\cdot)$ are independent and identically distributed (i.i.d.) random variables with common statistical parameter θ_i. Also for $i \neq i'$, $\texttt{msw}(i,\cdot,\cdot)$ and $\texttt{msw}(i',\cdot,\cdot)$ are independent.

We note that if $V_i = \{\texttt{1},\texttt{0}\}$, $\texttt{msw}(i,n,v)$ coincides with a *BS atom* $\texttt{bs}(i,n,v)$ in [9]. The second condition just says that the switch i takes a value v with probability $\theta_i(v)$.

2.3 A program example

For an example of a PRISM program, let us consider an HMM M whose possible states are $\{\texttt{s0},\texttt{s1}\}$, and whose possible output symbols are $\{\texttt{a},\texttt{b}\}$. Our HMM follows the definition described in [8] (not in [1]). The following is a program which represents M, where $\texttt{hmm}(\textit{String})$ denotes the observable fact that *String* is a string sampled from M. The probabilistic behaviors of state transition and output are specified by the switches of the form $\texttt{msw}(\texttt{tr}(\cdot),\cdot,\cdot)$ and $\texttt{msw}(\texttt{out}(\cdot),\cdot,\cdot)$, respectively. The length of a string is fixed to three.

```
%----------------------- Declarations -----------------------%
target(hmm,1).          % Only hmm(_) is observable.
data('hmm.dat').        % Data are contained in 'hmm.dat'.
values(init,[s0,s1]).   % Switch 'init' takes 's0' or 's1'.
values(tr(_),[s0,s1]). % Switch 'tr(_)' takes 's0' or 's1'.
values(out(_),[a,b]).   % Switch 'out(_)' takes 'a' or 'b'.
%-------------------------- Model ---------------------------%
strlen(3).              % The length of a string is fixed to 3.

hmm(Cs):- msw(init,null,Si),hmm(1,Si,Cs). % Start from state Si.
hmm(T,S,[C|Cs]):- strlen(L),T=<L,          % 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).                     %   Repeat above (recursion).
hmm(T,_,[]):- strlen(L),T>L.               % Finish the loop.
```

2.4 Learning PRISM programs

Learning a PRISM program means MLE (maximum likelihood estimation) of parameters in the program. That is, given observations G_t $(1 \leq t \leq T)$, we maximize the likelihood of these atoms $\prod_{t=1}^{T} P_{DB}(G_t{=}1|\theta)$ by adjusting parameters θ associated with msws in the program. The learning will be done by the following two-phase procedure:

1. Search exhaustively for S such that $S \in \psi_{DB}(G_t)$, for each observation G_t.
2. Run the EM algorithm and get the estimate of parameters θ.

In the second step, the EM algorithm makes calculations based on the statistics from $\psi_{DB}(G_t)$. This section describes a detail of the second step.

In what follows, we consider a set X of random variables as a random vector whose elements are X. Note that the realization of X is also a vector. Also i

(resp. $\dot{0}$) is used to denote a vector consisting of all 1s (resp. 0s). We now make some definitions for $G \in G^{patt}$. $relv(G)$ is a set of relevant switches to G, namely $\bigcup_{S \in \psi_{DB}(G)} S$. Then define

$$\Sigma_{DB}(G) \stackrel{\text{def}}{=} relv(G) \cup \left\{ \texttt{msw}(i,n,v) \;\middle|\; v \in V_i, \exists v' \left(\texttt{msw}(i,n,v') \in relv(G), v' \neq v \right) \right\},$$

and $\Sigma_{DB} \stackrel{\text{def}}{=} \bigcup_{G \in G^{patt}} \Sigma_{DB}(G)$. For $S \in \psi_{DB}(G)$, put

$$S^- \stackrel{\text{def}}{=} \left\{ \texttt{msw}(i,n,v) \mid v \in V_i, \exists v' (\texttt{msw}(i,n,v') \in S, v' \neq v) \right\}$$
$$S_{rest} \stackrel{\text{def}}{=} \Sigma_{DB} - (S \cup S^-)$$

S^- can be seen as the *complement* of S, since $S^- = \dot{0}$ if $S = \dot{1}$. I and θ respectively stand for the set of all switch names and the set of all parameters appearing in Σ_{DB}, i.e. $I \stackrel{\text{def}}{=} \{i \mid \texttt{msw}(i,\cdot,\cdot) \in \Sigma_{DB}\}$ and $\theta \stackrel{\text{def}}{=} \{\theta_i(v) \mid i \in I, v \in V_i\}$ in notation. Let S be an arbitrary subset of Σ_{DB}. S is said to be *inconsistent* if there are msws such that $\texttt{msw}(i,n,v)$, $\texttt{msw}(i,n,v') \in S$ and $v \neq v'$. If there is no such pair, S is *consistent*. Suppose $S, S' \subseteq \Sigma_{DB}$ are consistent respectively but $S \cup S'$ is inconsistent. In such a case, they are called *disjoint*. We assume the following *disjointness condition*:

Disjointness condition: *For any $S \in \psi_{DB}(G)$ and $G \in G^{patt}$, S is consistent, and disjoint from S' such that $S' \in \psi_{DB}(G), S' \neq S$.*

Under the disjointness condition and P_F's third condition, i.e., if $i \neq i'$ or $n \neq n'$ (i, i', n, and n' are all ground terms), random variables $\texttt{msw}(i,n,\cdot)$ and $\texttt{msw}(i',n',\cdot)$ are independent of each other, the likelihood of $G \in G^{patt}$ is calculated by the followings:

$$P_{DB}(G=1|\theta) = \textstyle\sum_{S \in \psi_{DB}(G)} P_F(S=\dot{1}|\theta) \quad (1)$$
$$P_F(S=\dot{1}|\theta) = \textstyle\prod_{i \in I, v \in V_i} \theta_i(v)^{\sigma_{i,v}(S)}, \quad (2)$$

where $\sigma_{i,v}(S)$ is the count of distinct $\texttt{msw}(i,\cdot,v)$s in the support set S. Under the notation in [9], $\sigma_{i,v}(S)$ corresponds to $|S \stackrel{i}{=} \dot{1}|_v$.

To realize MLE of PRISM program, we first introduce Q *function* by

$$Q(\theta_{new}, \theta) \stackrel{\text{def}}{=} \textstyle\sum_{t=1}^{T} \sum_x P_{DB}(\Sigma_{DB}=x|G_t=1,\theta) \log P_{DB}(\Sigma_{DB}=x, G_t=1|\theta_{new}). \quad (3)$$

From the definition, it is straightforward to show that $Q(\theta_{new},\theta) \geq Q(\theta,\theta)$ implies $\prod_{t=1}^{T} P_{DB}(G_t = 1|\theta_{new}) \geq \prod_{t=1}^{T} P_{DB}(G_t = 1|\theta)$. Our MLE procedure *learn-PRISM* shown below is an EM algorithm making use of this fact. It starts with initial values $\theta^{(0)}$, and iteratively updates, until saturation, $\theta^{(m)}$ to $\theta^{(m+1)}$ such that $Q(\theta^{(m+1)}, \theta^{(m)}) \geq Q(\theta^{(m)}, \theta^{(m)})$ to find θ that maximizes $\prod_{t=1}^{T} P_{DB}(G_t = \dot{1}|\theta)$.

procedure *learn-PRISM* **begin**
  foreach $i \in I, v \in V_i$ **do** /* *Initialize the parameters:* */
    Select some $\theta_i^{(0)}(v)$ such that $\sum_{v \in V_i} \theta_i^{(0)}(v) = 1$;

$\lambda^{(0)} := \sum_{t=1}^{T} \log P_{DB}(G_t = 1|\theta^{(0)})$;
$m := 0$;
repeat
  foreach $i \in I, v \in V_i$ **do begin** /* *E(xpectation)-step* */
    for $t := 1$ **to** T **do**
      $\widetilde{ON}_i^t(v) := \sum_{S \in \psi_{DB}(G_t)} P_F(S = i|\theta^{(m)}) \{\sigma_{i,v}(S) + \gamma_i(S)\theta_i^{(m)}(v)\}$;
    $ON_i(v) := \widetilde{ON}_i^t(v) / P_{DB}(G_t = 1|\theta^{(m)})$;
  end;
  $m := m + 1$;
  foreach i, v **do** $\theta_i^{(m)}(v) := ON_i(v) / \sum_{v' \in V_i} ON_i(v')$; /* *M(aximization)-step* */
  $\lambda^{(m)} := \sum_{t=1}^{T} \log P_{DB}(G_t = 1|\theta^{(m)})$;
until $\lambda^{(m)} - \lambda^{(m-1)} < \varepsilon$; /* *Terminate if the log-likelihood saturates.* */
end.

In *learn-PRISM*, ε is a small positive constant. By the definition of Σ_{DB} and S_{rest}, for $i \in I$, the count of distinct `msw(`i`,·,`v`)`s in S_{rest} is constant for each $v \in V_i$, so it is written as $\gamma_i(S)$. The above algorithm is a general learning algorithm applicable to any PRISM programs, but the existence of the term $\gamma_i(S)\theta_i(v)$ hinders efficient learning. We next replace P_F by a more specialized distribution to eliminate this term.

2.5 Constructing PRISM* programs

Suppose our program $DB = F \cup R$ with the basic distribution satisfies the following *uniqueness condition*.

Uniqueness condition: For any goal $G \in G^{patt}$, G's explanation does not explain other goals in G^{patt}. That is, for $G, G' \in G^{patt}$ and $G \neq G'$, $S \notin \psi_{DB}(G')$ if $S \in \psi_{DB}(G)$. Also $\sum_{G \in G^{patt}} P_{DB}(G = 1|\theta) = 1$ holds.

Then, under the disjoint and the uniqueness condition, it is possible to construct from DB a new program (let us call them PRISM* programs) $DB^* = F^* \cup R$ with a new fact set F^* and a new basic distribution P_{F^*} such that

1. Every atom in F^* takes the form of `msw(`i`,`n`,`v`)` and the range of v is $V_i^* \stackrel{\text{def}}{=} V_i \cup \{*\}$. Here V_i stands for a set of ground atoms assigned to i and $*$ is a new constant symbol appearing nowhere else.
2. Put $V_i^* = \{v_1, \ldots, v_K, *\}$. Exactly one of `msw(`$i$`,`$n$`,`$v_1$`)`, ..., `msw(`$i$`,`$n$`,`$v_K$`)`, `msw(`$i$`,`$n$`,*)` becomes true.
3. Define $S^* \stackrel{\text{def}}{=} \{$`msw(`$i$`,`$n$`,*)` $\mid$ `msw(`i`,`n`,·)` $\in S\}$, and $S_{rest}^* \stackrel{\text{def}}{=} \{$`msw(`$i$`,`$n$`,*)` $\mid$ `msw(`i`,`n`,·)` $\in S_{rest}\}$. For every $S \in \psi_{DB}(G), G \in G^{patt}$,

$$P_{F^*}(S = x, S^- = x^-, S^* = x^*, S_{rest} = z, S_{rest}^* = z^*|\theta) \stackrel{\text{def}}{=} \begin{cases} P_F(S = i|\theta) & \text{if } x = z^* = i, x^- = x^* = z = \dot{0} \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

From the disjointness and the uniqueness condition, it follows that P_{F^*} becomes a probability distribution. We also note that, in the third condition, P_{F^*} is defined on the original PRISM program's P_F and ψ_{DB}.

2.6 Learning PRISM* programs

The EM learning algorithm *learn-PRISM** for PRISM* programs is much simpler thanks to the change of the basic distribution. Define $\Sigma^*_{DB} \stackrel{\text{def}}{=} \Sigma_{DB} \cup \{\, \mathtt{msw}(i,n,*) \mid \mathtt{msw}(i,n,\cdot) \in \Sigma_{DB} \,\}$ and a new Q function (called Q^* *function*):

$$Q^*(\theta_{new}, \theta) \stackrel{\text{def}}{=} \sum_{t=1}^{T} \sum_x P_{\text{DB}^*}(\Sigma^*_{DB} = x | G_t = 1, \theta) \log P_{\text{DB}^*}(\Sigma^*_{DB} = x, G_t = 1 | \theta_{new}). \tag{5}$$

Similarly to *learn-PRISM*, we can derive *learn-PRISM** which updates $\theta^{(m)}$ to $\theta^{(m+1)}$ such that $Q^*(\theta^{(m+1)}, \theta^{(m)}) \geq Q^*(\theta^{(m)}, \theta^{(m)})$, and thus realizes MLE for a PRISM* program. By definition of PRISM*, it is easy to show $P_{DB}(G = 1|\theta) = P_{\text{DB}^*}(G = 1|\theta)$ for any θ and $G \in G^{patt}$, hence we can say *learn-PRISM** also realizes MLE for the original PRISM program. Also, it is roughly proved in Appendix A.2 that by definition of PRISM*, *learn-PRISM** is just *learn-PRISM* with the term $\gamma_i(S)\theta_i(v)$ deleted which causes computational inefficiency.

3 Graphical EM algorithm

This section introduces another learning procedure for PRISM programs. The new procedure makes use of a graphical structure for efficient learning.

1. For each observation G_t, We first have an exhaustive search for S such that $S \in \psi_{DB}(G_t)$. While searching, using search techniques such as tabulation, we construct a data structure, called a *support graph*, for each G_t.
2. We then run a graph-based EM algorithm, called the *graphical EM algorithm*, on the constructed support graphs, and get the estimate of parameters θ.

In the rest of this section, we assume that the suitable support graphs are given, and consider the second step. It then can be proved that the learning algorithm for PRISM* programs (i.e., *learn-PRISM**) and the graphical EM algorithm yield the same estimate of θ (see Appendix B).

3.1 Support graphs

The support graph for G_t $(1 \leq t \leq T)$ is a triplet (U_t, E_t, l_t), where U_t is a set of nodes, $E_t \subseteq U_t \times U_t$ is a set of edges, and $l_t : U_t \to (\Sigma^t_{DB} \cup \{G_t, \square\})$ is called a *labeling function*. A support graph (U_t, E_t, l_t) for G_t should satisfy the following conditions:

- (U_t, E_t) is a directed acyclic graph which has exactly one node (referred to as u_t^{top}) that is not terminal node of any edges in E_t, and exactly one node (referred to as u_t^{bot}) that is not initial node of any edges in E_t. We define $U_t^{body} \stackrel{\text{def}}{=} U_t - \{u_t^{top}, u_t^{bot}\}$.
- u_t^{top} and u_t^{bot} have special labels, i.e., $l_t(u_t^{top}) = G_t$ and $l_t(u_t^{bot}) = \square$.
- In any path from u_t^{top} to u_t^{bot}, no switch occurs more than once, i.e., $u, u' \in \mathit{nodes}(r) \wedge u \neq u' \Rightarrow l_t(u) \neq l_t(u')$ for $r \in \mathit{path}(u_t^{top}, u_t^{bot})$, where $\mathit{path}(u, u')$ is a set of directed paths from u to u', $\mathit{nodes}(r)$ is a set of nodes in the directed path r.

- For $r, r' \in path(u_t^{top}, u_t^{bot})$ and $r \neq r'$, $labels(r)$ and $labels(r')$ are disjoint from each other, where $labels(r) \stackrel{\text{def}}{=} \{l_t(u) \mid u \in nodes(r)\} - \{G_t, \square\}$ for the directed path r in (U_t, E_t, l_t).
- $\psi_{DB}(G_t) = \{\pi^{l_t} \mid \pi \in \Pi^{(U_t,E_t)}\}$, where

$$\Pi^{(U_t,E_t)}(u,u') \stackrel{\text{def}}{=} \left\{\pi \mid \exists r\big(r \in path(u,u'), \pi = nodes(r) - \{u_t^{top}, u_t^{bot}\}\big)\right\},$$
$$\Pi^{(U_t,E_t)} \stackrel{\text{def}}{=} \Pi^{(U_t,E_t)}(u_t^{top}, u_t^{bot}),$$
$$\pi^{l_t} \stackrel{\text{def}}{=} \{l_t(u) \mid u \in \pi\} \text{ for } \pi \subseteq U_t^{body}.$$

We sometimes omit the superscript (U_t, E_t) and abbreviate π^{l_t} as π^l if it does not make a confusion.

3.2 Graphical EM algorithm

Once we have constructed the support graphs for all observations, parameters are learned by the graphical EM algorithm. To specify this, we should add some definitions. *parent*(u) and *child*(u) refer to a set of parent nodes and that of child nodes of u, respectively. $p(u, \theta)$ is then defined for $u \in \bigcup_t U_t$ and θ:

$$p(u,\theta) \stackrel{\text{def}}{=} \begin{cases} \theta_i(v) & \text{if } u \in U_t^{body} \text{ and } l_t(u) = \texttt{msw}(i,\cdot,v) \\ 1 & \text{otherwise} \end{cases}$$

The graphical EM algorithm consists of a procedure *learn-gEM* and a function *forward-backward*. We prepare variables $\alpha(u)$ and $\beta(u)$ for each $u \in U_t, 1 \leq t \leq T$, called *forward probability* and *backward probability* of node u, respectively.

procedure *learn-gEM* **begin**
  foreach $i \in I, v \in V_i$ **do** /* *Initialize the parameters:* */
    Select some $\theta_i^{(0)}(v) \in (0,1)$ such that $\sum_v \theta_i^{(0)}(v) = 1$;
  for $t := 1$ **to** T **do** $P_t :=$ *forward-backward*$(U_t, E_t, l_t, \theta^{(0)})$;
  $\lambda^{(0)} := \sum_{t=1}^{T} \log P_t$;
  $m := 0$;
  repeat
    foreach i, v **do** $on_i(v) := 0$;
    for $t := 1$ **to** T **do begin**
      $s := U_t^{body}$;
      foreach i, v **do** $\widetilde{on}_i^t(v) := 0$;
      while $s \neq \emptyset$ **do begin** /* *E-step:* */
        Choose some u from s;
        if $l_t(u) = \texttt{msw}(i,\cdot,v)$ **then** $\widetilde{on}_i^t(v) := \widetilde{on}_i^t(v) + \alpha(u)\beta(u)$;
        $s := s - \{u\}$;
      end;
      foreach i, v **do** $on_i(v) := on_i(v) + \widetilde{on}_i^t(v)/P_t$;
    end;
    $m := m + 1$;
    foreach i, v **do** $\theta_i^{(m)}(v) := on_i(v)/\sum_{v' \in V_i} on_i(v')$; /* *M-step* */
    for $t := 1$ **to** T **do** $P_t :=$ *forward-backward*$(U_t, E_t, l_t, \theta^{(m)})$;
    $\lambda^{(m)} := \sum_{t=1}^{T} \log P_t$;
  until $\lambda^{(m)} - \lambda^{(m-1)} < \varepsilon$; /* *Terminate if the log-likelihood saturates.* */
end.

function *forward-backward* (U_t, E_t, l_t, θ) **begin**
  foreach $u \in U_t$ **do begin**
    $\alpha(u) :=$ **undef**; $\beta(u) :=$ **undef**;
  end;
  $\alpha(u_t^{top}) := 1$; $\beta(u_t^{bot}) := 1$; $s := child(u_t^{top})$; $s' := parent(u_t^{bot})$;
  while $s \neq \emptyset$ **do begin** /* *Calculate forward probabilities for each node:* */
    Choose some u from s such that $\forall u' \in parent(u)\big(\ \alpha(u') \neq$ **undef** $\big)$;
    $\alpha(u) := \Big(\sum_{u' \in parent(u)} \alpha(u')\Big) p(u, \theta)$;
    $s := (s \cup child(u)) - \{u\}$;
  end;
  while $s' \neq \emptyset$ **do begin** /* *Calculate backward probabilities for each node:* */
    Choose some u from s' such that $\forall u' \in child(u)\big(\ \beta(u') \neq$ **undef** $\big)$;
    $\beta(u) := \sum_{u' \in child(u)} \beta(u') p(u', \theta)$;
    $s' := (s' \cup child(u)) - \{u\}$;
  end;
  return $\alpha(u_t^{bot})$; /* *Return the likelihood.* */
end.

3.3 Learning HMMs

Let us consider again the program in Section 2.3. Fig. 1 illustrates the support graph of an observed goal `hmm([a,b,a])`. Each node in the graph is labeled with `msw(·,·,·)`, `hmm([a,b,a])`, or □. Given such a support graph (U_t, E_t, l_t), *learn-gEM* and *forward-backward* takes $O(|E_t|)$ of calculations in each **repeat** loop. So, letting N be the number of states, and L the length of a string, the graphical EM algorithm takes $O(N^2L)$ of calculations for each parameter updating. As described in [8], Baum-Welch algorithm also takes $O(N^2L)$, so it is concluded that the graphical EM algorithm can simulate Baum-Welch algorithm via the HMM written as a PRISM program.

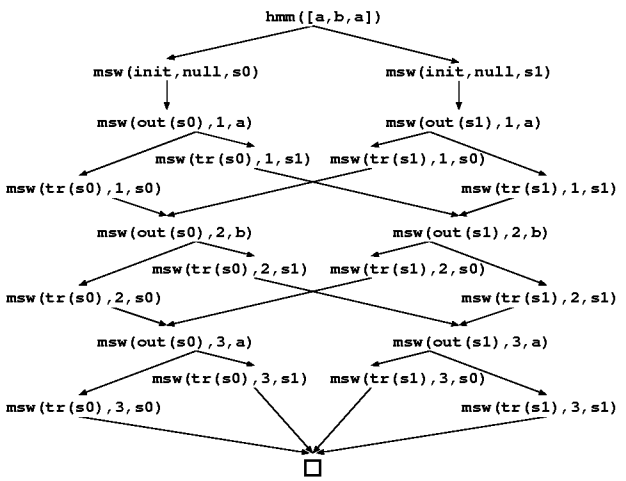

Fig. 1. The support graph with the goal `hmm([a,b,a])`.

4 Related works

So far, many probabilistic extensions of logic programs have been proposed. We here mention some of related works briefly (In [11], we have also mentioned other related works such as [4, 7]). Muggleton's *stochastic logic programs* (SLPs) [5] combine probabilities with first-order logic programs, but no mention is made about the parameter learning. In Riezler's probabilistic constraint logic programming [12] and Cussens's loglinear models using SLPs [3], the probability distribution (a loglinear distribution) is defined on the set Ω of *proof trees*. It is of the form $Pr(\omega) = Z^{-1} \exp(\sum_i \lambda_i f_i(\omega))$ for $\omega \in \Omega$, where each f_i is the feature of a proof tree and λ_i is the parameter. The problem in their framework is that the adoption of Ω as a sample space could be an obstacle to the logical or procedural treatment of negation. Besides, the calculation of a normalizing constant Z is generally intractable, and Riezler proposed no polynomial learning algorithm for a specialized class of models such as HMMs. In contrast, in PRISM modeling, we can enjoy efficient learning as shown in this paper, though users must write programs so that probabilities of all observable ground atoms sum up to one, which eliminates the needs of normalization. There seems to be a trade-off between the efficiency of learning and the burden of programming on the user.

5 Discussion

We have presented a new framework for the modeling language PRISM, in which efficient parameter learning is achieved for a certain class of symbolic-statistical models. We showed that, if the objective model can be represented by a PRISM program satisfying two conditions (the disjointness and the uniqueness condition), then we can construct the corresponding PRISM* program for which an efficient learning is possible. A user has only to write programs so that these two conditions are met. In reality, we can say that they are not too restrictive in the sense that PRISM programs which represent widely-known symbolic-statistical models like HMMs, BNs, and PCFGs satisfy these conditions.

We then presented a graph-based EM algorithm (the graphical EM algorithm) which runs on a graphical data structure (a support graph) of a PRISM program for each observation. We have roughly shown that, for a given PRISM program, the EM algorithm for the (automatically-constructed) PRISM* program and the graphical EM, given appropriate support graphs, yield the same estimate. This justifies to use the graphical EM instead of the old learning algorithm for PRISM programs used so far, which is general but computationally inefficient. We also showed that, if appropriate support graphs are given for an HMM, the graphical EM only requires the same computational complexity as Baum-Welch algorithm. This implies that the graphical EM is a generalization of Baum-Welch. Similarly, it is straightforward to show the algorithm which finds the most likely minimal support set (or the most likely *explanation*) for an observation. The algorithm, omitted due to the space limitation, can be considered as a generalization of Viterbi algorithm [1, 8], also developed for HMMs.

There remains a lot to be done for our new framework. Generating such a support graph is still a problem, so we are currently planning to adopt search

techniques such as tabulation. We also need to study of the computational relationship between the graphical EM and other specialized EM algorithms, e.g., Inside-Outside algorithm [1] for PCFGs.

References

1. Charniak, E., *Statistical Language Learning*, The MIT Press, 1993.
2. Clark, K., Negation as failure, In Gallaire, H., and Minker, J. (eds), *Logic and Databases*, pp.293–322, Plenum Press, 1978.
3. Cussens, J., Loglinear models for first-order probabilistic reasoning, *Proc. of the 15th Conf. on Uncertainty in Artificial Intelligence*, 1999.
4. Koller, D., and Pfeffer A., Learning probabilities for noisy first-order rules. *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pp.1316–1321, 1997.
5. Muggleton, S., Stochastic logic programs, L. De Raedt (ed.) *Advances in Inductive Logic Programming*, IOS Press, pp.254–264, 1996.
6. Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.
7. Poole, D., Probabilistic Horn abduction and Bayesian networks, *Artificial Intelligence*, Vol.64, pp.81–129, 1993.
8. Rabiner, L. R., A tutorial on hidden Markov models and selected applications in speech recognition, *Proc. of the IEEE*, Vol.77, No.2, 1989.
9. Sato, T., A statistical learning method for logic programs with distribution semantics, *Proc. of the 12th Intl. Conf. of Logic Programming*, pp.715–729, 1995.
10. Sato, T., and Kameya, Y., PRISM: a symbolic-statistical modeling language, *Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pp.1330–1335, 1997.
11. Sato, T., Modeling scientific theories as PRISM programs, *ECAI-98 Workshop on Machine Discovery*, pp.37–45, 1998.
12. Riezler, S., Probabilistic constraint logic programming, Arbeitsberichte des SFB 340 Bericht Nr. 117, Universität Tübingen. 1997.
13. Tanner, M., *Tools for Statistical Inference* (2nd ed.), Springer-Verlag, 1993.

A Derivation of *learn-PRISM* and *learn-PRISM**

A.1 *learn-PRISM*

To derive *learn-PRISM*, we transform Q function. We hereafter consider $0 \log 0 = 0$, and abbreviate $P_{DB}(\Sigma_{DB}=x,\ldots)$ as $P_{DB}(x,\ldots)$ and $P_{DB}(G_t=1|\theta)$ as P_t.

$$\begin{aligned}
Q(\theta_{new},\theta) &= \sum_{t=1}^{T} \tfrac{1}{P_t} \sum_x P_{DB}(x, G_t=1|\theta) \log P_{DB}(x, G_t=1|\theta_{new}) \\
&= \sum_t \tfrac{1}{P_t} \sum_{S\in\psi_{DB}(G_t)} \sum_z P_{DB}(S=\dot{1}, S^-=\dot{0}, S_{rest}=z, G_t=1|\theta) \cdot \\
&\qquad\qquad \log P_{DB}(S=\dot{1}, S^-=\dot{0}, S_{rest}=z, G_t=1|\theta_{new}) \\
&= \sum_t \tfrac{1}{P_t} \sum_S \sum_z P_F(S=\dot{1}, S_{rest}=z|\theta) \log P_F(S=\dot{1}, S_{rest}=z|\theta_{new}).
\end{aligned}$$

In addition, under the condition of P_F, for each $S \in \psi_{DB}(G_t)$ $(1 \le t \le T)$, S and S_{rest} are independent of each other. So, we have

$$\begin{aligned}
Q(\theta_{new},\theta) = \sum_t \tfrac{1}{P_t} \sum_S P_F(S=\dot{1}|\theta) \cdot \\
\left\{ \log P_F(S=\dot{1}|\theta_{new}) + \sum_z P_F(S_{rest}=z|\theta) \log P_F(S_{rest}=z|\theta_{new}) \right\} \quad (6)
\end{aligned}$$

Let $|S_{rest} \overset{i}{=} z|_v$ be the count of equations $\mathtt{msw}(i,\cdot,v) = 1$ which occurs in $S_{rest} = z$. Then, similarly to Eq. 2, $P_F(S_{rest} = z|\theta_{new}) = \prod_{i,v} \theta_i'(v)^{|S_{rest} \overset{i}{=} z|_v}$ holds. Substituting this and Eq. 2 into Eq. 6, the content of $\{\cdot\}$ in Eq. 6 becomes:

$$\begin{aligned}&\textstyle\sum_{i\in I, v\in V_i} \sigma_{i,v}(S) \log \theta_i'(v) + \sum_z P_F(S_{rest}=z|\theta) \sum_{i\in I, v\in V_i} |S_{rest} \overset{i}{=} z|_v \log \theta_i'(v)\\ &= \textstyle\sum_{i,v} \big(\sigma_{i,v}(S) + \sum_z P_F(S_{rest}=z|\theta)|S_{rest} \overset{i}{=} z|_v\big) \log \theta_i'(v). \end{aligned} \tag{7}$$

Let $S_{rest}^{i,v}$ denote a set of $\mathtt{msw}(i,\cdot,v)$s included in S_{rest} (S_{rest} will be divided to the disjoint sets $S_{rest}^{i,v}$ $(i \in I, v \in V_i)$), and let $z_{i,v}$ denote the realization of $S_{rest}^{i,v}$. Then, from the fact that if $i \neq i'$ or $v \neq v'$ then $|S_{rest}^{i',v'} \overset{i}{=} z_{i',v'}|_v = 0$, and that distinct $S_{rest}^{i,v}$ $(i \in I, v \in V_i)$ are independent of each other, the following holds:

$$\textstyle\sum_z P_F(S_{rest}=z|\theta)|S_{rest} \overset{i}{=} z|_v = \sum_{z_{i,v}} P_F(S_{rest}^{i,v}=z_{i,v}|\theta)|S_{rest}^{i,v} \overset{i}{=} z_{i,v}|_v.$$

The right-hand side can be considered as the expectation of the count of switch i taking v, in which switch i is sampled for $\gamma_i(S)$ times under the parameter θ_i, so it equals $\gamma_i(S)\theta_i(v)$. That is, $\sum_z P_F(S_{rest}=z|\theta)|S_{rest} \overset{i}{=} z|_v = \gamma_i(S)\theta_i(v)$ holds. Here we get the following inequality:

$$\begin{aligned} Q(\theta_{new},\theta) &= \textstyle\sum_{t=1}^T \frac{1}{P_t} \sum_S P_F(S=\dot{1}|\theta) \sum_{i,v} \Big(\sigma_{i,v}(S) + \gamma_i(S)\theta_i(v)\Big) \log \theta_i'(v)\\ &= \textstyle\sum_{i,v} \Big(\sum_t \frac{1}{P_t} \sum_S P_F(S=\dot{1}|\theta)\Big(\sigma_{i,v}(S) + \gamma_i(S)\theta_i(v)\Big)\Big) \log \theta_i'(v)\\ &= \textstyle\sum_{i\in I, v\in V_i} ON_i(v,\theta) \log \theta_i'(v) \;\leq\; \sum_{i,v} ON_i(v,\theta) \log \frac{ON_i(v,\theta)}{\sum_{v'\in V_i} ON_i(v',\theta)}, \end{aligned}$$

where $ON_i(v,\theta) \overset{\text{def}}{=} \sum_{t=1}^T \frac{1}{P_t} \sum_{S\in\psi_{DB}(G_t)} P_F(S=\dot{1}|\theta)\{\sigma_{i,v}(S) + \gamma_i(S)\theta_i(v)\}$. Replacing θ and θ_{new} by $\theta^{(m)}$ and $\theta^{(m+1)}$, respectively, the update $\theta_i^{(m+1)}(v) := ON_i(v,\theta^{(m)}) / \sum_{v'\in V_i} ON_i(v',\theta^{(m)})$ yields $Q(\theta^{(m+1)},\theta^{(m)}) \geq Q(\theta^{(m)},\theta^{(m)})$, so this update of θ realizes MLE of the PRISM program. Here it is obvious that *learn-PRISM* realizes the above procedure. □

A.2 *learn-PRISM**

Similarly to the derivation of *learn-PRISM*, we proceed to transform Q^* function. Due to the space limitation, we abbreviate S_{rest} as S_r, and note that $P_{\mathrm{DB}^*}(G_t=1|\theta) = P_{DB}(G_t=1|\theta)$ $(= P_t)$.

$$\begin{aligned} &Q^*(\theta_{new},\theta)\\ &= \textstyle\sum_{t=1}^T \frac{1}{P_t} \sum_x P_{\mathrm{DB}^*}(\Sigma_{DB}^*=x, G_t=1|\theta) \log P_{\mathrm{DB}^*}(\Sigma_{DB}^*=x, G_t=1|\theta_{new})\\ &= \textstyle\sum_{t=1}^T \frac{1}{P_t} \sum_{S\in\psi_{DB}(G_t)} P_{\mathrm{DB}^*}(S=\dot{1}, S^-=\dot{0}, S^*=\dot{0}, S_r=\dot{0}, S_r^*=\dot{1}, G_t=1|\theta) \cdot\\ &\qquad\qquad \log P_{\mathrm{DB}^*}(S=\dot{1}, S^-=\dot{0}, S^*=\dot{0}, S_r=\dot{0}, S_r^*=\dot{1}, G_t=1|\theta_{new})\\ &= \textstyle\sum_{t=1}^T \frac{1}{P_t} \sum_S P_{F^*}(S=\dot{1}, S^-=\dot{0}, S^*=\dot{0}, S_r=\dot{0}, S_r^*=\dot{1}|\theta) \cdot\\ &\qquad\qquad \log P_{F^*}(S=\dot{1}, S^-=\dot{0}, S^*=\dot{0}, S_r=\dot{0}, S_r^*=\dot{1}|\theta_{new})\\ &= \textstyle\sum_{t=1}^T \frac{1}{P_t} \sum_{S\in\psi_{DB}(G_t)} P_F(S=\dot{1}|\theta) \log P_F(S=\dot{1}|\theta_{new}). \end{aligned} \tag{8}$$

Note that the right-hand side of Eq. 8 equals to that of Eq. 6 with a term $\sum_z P_F(S_{rest}=z \mid \theta) \log P_F(S_{rest}=z|\theta_{new}) = 0$. As described in the previous section, this term results in $\gamma_i(S)\theta_i(v)$, so the obtained algorithm, *learn-PRISM**, is just *learn-PRISM* with $\gamma_i(S)\theta_i(v) = 0$. □

B Equivalence of *learn-PRISM** and *learn-gEM*

In this section, we present an outline of the proof of equivalence of *learn-PRISM** and *learn-gEM*. We first consider the support graph (U_t, E_t, l_t) for the goal G_t. After executing *forward-backward*(U_t, E_t, l_t, θ), the followings hold for $u \in U_t$:

$$\alpha(u) = \begin{cases} 1 & \text{if } u = u_t^{top} \\ p(u,\theta) & \text{if } u \in \mathit{child}(u_t^{top}) \\ \left(\sum_{\pi \in \Pi_f(u)} P_F(\pi^l = \dot{1}|\theta)\right) p(u,\theta) & \text{otherwise,} \end{cases} \tag{9}$$

$$\beta(u) = \begin{cases} 1 & \text{if } u = u_t^{bot} \text{ or } u \in \mathit{parent}(u_t^{bot}) \\ \sum_{\pi \in \Pi_b(u)} P_F(\pi^l = \dot{1}|\theta) & \text{otherwise,} \end{cases} \tag{10}$$

$$\alpha(u)\beta(u) = \sum_{\pi \in \Pi, u \in \pi} P_F(\pi^l = \dot{1}|\theta), \tag{11}$$

where $\Pi_f^{(U_t,E_t)}(u) \stackrel{\text{def}}{=} \Pi^{(U_t,E_t)}(u_t^{top}, u)$, and $\Pi_b^{(U_t,E_t)}(u) \stackrel{\text{def}}{=} \Pi^{(U_t,E_t)}(u, u_t^{bot})$. The proof is done by induction on the structure of (U_t, E_t, l_t).

Then, the return value of *forward-backward*(U_t, E_t, l_t, θ) is equal to the likelihood $P_{DB}(G_t=1|\theta)$, because, from definition of support graphs and $\alpha(u)$ above,

$$\begin{aligned} \alpha(u_t^{bot}) &= \left(\sum_{\pi \in \Pi_f(u_t^{bot})} P_F(\pi^l = \dot{1}|\theta)\right) \underbrace{p(u_t^{bot}, \theta)}_{=1} \\ &= \sum_{\pi \in \Pi} P_F(\pi^l = \dot{1}|\theta) \quad = \sum_{S \in \psi_{DB}(G_t)} P_F(S = \dot{1}|\theta) = P_{DB}(G_t=1|\theta). \end{aligned}$$

To show that *learn-PRISM* and *learn-gEM* starting from the same initial parameters $\theta^{(0)}$ yields the same estimate, it is enough to show that, just before parameter updating, the value of $\widetilde{ON}_i^t(v)$ in *learn-PRISM* and $\widetilde{on}_i^t(v)$ in *learn-gEM* are equal for $1 \le t \le T, i \in I, v \in V_i$.

$$\begin{aligned} \widetilde{on}_i^t(v) &= \sum_{u \in U_t, l_t(u) = \mathtt{msw}(i,\cdot,v)} \alpha(u)\beta(u) \quad \left(\text{from } \textbf{while} \text{ loop of } \textit{learn-gEM}.\right) \\ &= \sum_{u \in U_t, l_t(u) = \mathtt{msw}(i,\cdot,v)} \sum_{\pi \in \Pi, u \in \pi} P_F(\pi^l = \dot{1}|\theta) \\ &= \sum_{u \in U_t, l_t(u) = \mathtt{msw}(i,\cdot,v), \pi \in \Pi, u \in \pi} P_F(\pi^l = \dot{1}|\theta) \quad = \sum_{\pi \in \Pi} \sum_{l_t(u) = \mathtt{msw}(i,\cdot,v), u \in \pi} P_F(\pi^l = \dot{1}|\theta) \\ &= \sum_{\pi \in \Pi} P_F(\pi^l = \dot{1}|\theta) \underbrace{\sum_{l_t(u) = \mathtt{msw}(i,\cdot,v), u \in \pi} 1}_{=\sigma_{i,v}(S)} = \sum_{S \in \psi_{DB}(G_t)} P_F(S = \dot{1}|\theta)\sigma_{i,v}(S) = \widetilde{ON}_i^t(v). \end{aligned}$$

□

This article was processed using the LaTeX macro package with LLNCS style