A Generic Approach to EM learning for Symbolic-statistical Models

Taisuke Sato SATO@MI.CS.TITECH.AC.JP
Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan

Abstract

We present a generic approach to EM learning (i.e. parameter learning by ML estimation for probabilistic models using the EM algorithm). We write domain-dependent programs in a symbolic-statistical modeling language PRISM to define distributions but apply a common EM algorithm to them. Differences in EM learning for each model are subsumed by differences in programs, and hence we have only to write PRISM programs to perform EM learning even for new probabilistic models and the rest of task is automatically carried out by the PRISM's learning routine.

1. Introduction

Parameter learning by ML (maximum likelihood) estimation for probabilistic models is one of the most basic techniques for machine learning and the EM algorithm (Dempster et al., 1977) has been a primary tool for ML estimation. So far however, EM learning, i.e. parameter learning by the EM algorithm has been carried out domain-dependently. In other words one has to use different EM algorithms for different applications or when no EM algorithm is available, he or she has to invent a new EM algorithm.

In this paper we present a different approach with the following features. First we use a single EM algorithm called the *gEM (graphical EM) algorithm* (Kameya & Sato, 2000) for every application regardless of whether it belongs to a known model or not. Second we use a symbolic-statistical modeling language PRISM[1] and write a program to define a parameterized distribution for each probabilistic model. To learn parameters of the distribution, we first apply the program to data and extract a certain AND-OR graph called *explanation graph* representing the likelihood function, and then run the gEM algorithm on it. So differences in distributions are reflected on differences in programs and do not affect the EM algorithm itself. Put it differently we do not need to invent a new EM algorithm even for a new model. Third computationally speaking, a kind of dynamic programming is built-in in PRISM and our approach can be reasonably competitive.[2]

We proved in (Sato & Kameya, 2001) that PRISM programs representing popular symbolic-statistical models, i.e. singly connected BNs (Bayesian networks), HMMs (hidden Markov models) and PCFGs (probabilistic context free grammars) have the same time complexity in EM learning as their specialized counterparts, i.e. EM learning by Pearl's belief propagation (Pearl, 1988), the Baum-Welch algorithm (Rabiner & Juang, 1993) and the Inside-Outside algorithm (Manning & Schütze, 1999) respectively. In addition in the case of PCFGs, it is experimentally confirmed that the gEM algorithm runs faster than the Inside-Outside algorithm by orders of magnitude provided explanation graphs are given (Sato & Kameya, 2001).

We have tested other known and unknown EM learning by PRISM programs in addition to these well-known models to see the strength and weakness of our approach. The list includes Naive Bayes, linkage analysis (Lander-Green algorithm (Kruglyak et al., 1996), Elston-Stewart algorithm (Elston & Stewart, 1971)), profile HMMs for alignment, generalized LR(k) parsing (Inui et al., 1997), game modeling and most recently left-corner parsing (Manning, 1997) and context free graph grammars (Rozenberg, 1997). What has become clear from these modeling experiences is that our approach fits well in the development stage of new statistical models because we can explore and test them at higher level without the additional trouble of implementing EM algorithms from scratch. Also continuing efforts for efficient implementation of PRISM

Appearing in *Proceedings of the 4th Learning Language in Logic Workshop (LLL05)*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

[1]URL = `http://sato-www.cs.titech.ac.jp/prism/`

[2]However we do not claim that it is as efficient as EM learning by a specialized EM algorithm implemented in C.

(Zhou et al., 2004) made it usable in such a stage.[3] In this paper, we show how to perform EM learning, using PRISM, for a stochastic left-corner parsing model, and two classes of stochastic context free graph grammars. We remark that there is little or no literature on EM learning of these probabilistic models.

In what follows, after reviewing PRISM briefly by a simple example, we look at each of the above mentioned EM learning. The reader is assumed to be familiar with the EM algorithm (McLachlan & Krishnan, 1997), logic programming (Doets, 1994) and stochastic natural language processing (Manning & Schütze, 1999).

2. Symbolic-statistical modeling

By symbolic-statistical modeling we mean modeling of a distribution over structured objects represented by symbols. Typical symbolic-statistical models are discrete BNs, HMMs and PCFGs where BNs define distributions over symbols, HMMs over strings and PCFGs over parse trees. They have been developed as different formalisms and people have to write code for different EM algorithms. PRISM breaks this order-made approach (Sato & Kameya, 1997; Sato & Kameya, 2001). It offers a generic language for describing symbolic-statistical models together with a unified EM learning routine. As a programming language it is an augmentation of Prolog by probabilistic built-ins for calculating and learning probabilities. So the user writes just Prolog programs using some probabilistic predicates. However the semantics, *distribution semantics* (Sato, 1995), is significantly extended compared to the standard least model semantics for logic programs in which a program denotes a (σ additive) probability measure over the set of possible Herbrand interpretations. The current PRISM[4] is implemented on top of B-Prolog using its tabling mechanism (Zhou et al., 2004). It can also deal with negation (failure), though we do not discuss this new feature in this paper.

We here explain a PRISM program for the sake of self-containedness. Figure 1 is a simple PRISM program modeling the inheritance of ABO blood types. `target(btype,1)` says we observe an event represented by an atom `btype(·)`. `values(abo,[a,b,o])` introduces a discrete random variable `abo` (i.i.d.s) whose range is $\{\mathtt{a},\mathtt{b},\mathtt{o}\}$ and the corresponding probabilities are set by `set_sw/2` to 0.3, 0.3 and 0.4 respectively on loading the program.

```
target(btype,1).
values(abo,[a,b,o]).
:- set_sw(abo,0.3+0.3+0.4).

btype(X):- pg_table(X,[Gf,Gm]),gtype(Gf,Gm).
pg_table(X,GT):-
  ((X=a;X=b),(GT=[X,o];GT=[o,X]; GT=[X,X])
  ; X=o, GT=[o,o]
  ; X=ab, (GT=[a,b];GT=[b,a]) ).
gtype(Gf,Gm):- msw(abo,Gf),msw(abo,Gm).
```

Figure 1. ABO-blood type program

The following clauses can be read just like an ordinary Prolog program. The only difference from Prolog is the use of `msw/2` predicate in the definition of `gtype` clause. In general when a random variable named 'n' is introduced by `values(`n`,[`$v_1,\ldots,v_k$`])` and its distribution is specified by `set_sw/2` as `set_sw(n,[`$p_1+\cdots+p_k$`])` ($\sum_i p_i = 1$), `msw(`n`,Y)` probabilistically returns in `Y` one of $\{v_1,\ldots,v_k\}$ with $P(n = v_i) = p_i$. These probabilities are called *parameters*. We assume different occurrences of the same random variable name denote i.i.d.s.

The purpose of this program is to estimate parameters for the distribution of `abo` genes from observations given as a list like `[btype(a),btype(ab),`$\cdots$`]`. In view of statistical modeling, `btype(·)` is an incomplete data with `abo` as a hidden variable. So the EM algorithm applies to this parameter estimation problem. The unique feature of our approach is the existence of a search phase that logically reduces an observed atom to a disjunction of explanations[5] represented as an AND-OR graph called *explanation graph* to which a common EM algorithm, the graphical EM algorithm, is applied. The formula below shows the explanation graph for `btype(a)`.

```
btype(a) <=>
    gtype(a,o) v gtype(o,a) v gtype(a,a)
gtype(a,o) <=> msw(abo,o) & msw(abo,a)
gtype(o,a) <=> msw(abo,a) & msw(abo,o)
gtype(a,a) <=> msw(abo,a) & msw(abo,a)
```

As the search phase is carried out using tabling (memoizing), the generated explanation graph generally

[3]For the record in the case of PCFGs, EM learning for a PCFG of moderate size (860 production rules) using a real corpus, ATR corpus(Uratani et al., 1994), containing 11000 sentences can now be completed in less than 10 minutes on a PC with 2MHz CPU.

[4]URL = `http://sato-www.cs.titech.ac.jp/prism/`

[5]An explanation is a conjunction of `msw` atoms.

shares subgraphs hierarchically, and this hierarchical structure sharing makes it possible to compute probabilities in a dynamic programming manner.

3. EM Learning for Probabilistic Left Corner Grammars

3.1. Probabilistic LC grammars

In this section, we deal with EM learning for *PLCGs (probabilistic left-corner grammars)*. PLCGs construct parse trees in a bottom-up manner using CFG rules but with different parameterization from PCFGs. In a PCFG probabilities $P(A \to \alpha \mid A)$ in top-down rule selection of $A \to \alpha$ given a non-terminal A are taken as parameters while in a PLCG projection probabilities $P(A \to B\beta \mid B, G)$ given a category B of a completed subtree and a goal category G, for instance are included in the parameter set for a PLCG. As rule selection is affected by the completed tree B, PLCGs are expected to be more context sensitive than PCFGs.

Although PCFGs and PLCGs use the same set of CFG rules, implementing EM learning for PLCGs seems much harder than PCFGs. In fact although there is some literature on PLCGs (Manning, 1997; Roark & Johnson, 1999), parameters there are obtained by counting using a tree bank, not by EM learning. The only literature we found that uses EM learning is (Van Uytsel et al., 2001) which describes a specialized EM algorithm derived for lexicalized PLCGs.

By contrast we do not need such a derivation. All one has to do is to write an efficient program while observing the *principle of generative modeling* which states that a model should describe a sequential process of generating an observable output and once a probabilistic choice is made, there must be no failure in the subsequent process (*failure-free condition*). This principle looks rigid but ensures the mathematical correctness of our statistical model.[6]

3.2. Three operations for PLCG parsing

We encode a PLC grammar as a PRISM program in Figure 2 which is a probabilistic extension of a non-probabilistic left-corner parser described in (Manning, 1997). The behavior of this program as a parser exactly follows the non-probabilistic LC parser except the use of msw/2. It parses a sentence of length N in $O(N^3)$ time thanks to the tabling (caching) mechanism of PRISM.

[6]Recently we succeeded in relaxing the failure-free condition to explore a wider class of distributions(Sato & Kameya, 2004).

So let us check how it works as a sentence generator. The top-goal for this purpose is :-plc(Ws) which, after invoking g_call/3 and lc_call/4 recursively, returns a list Ws = $[w_1, \ldots, w_k]$ of terminals as a grammatical sentence.

```
plc(Ws):-
   start_symbol(C), g_call([C],Ws,[]).

g_call([],L,L).     % shift
g_call([G|R],[Wd|L],L2):-
   ( terminal(G), G=Wd, L1=L
   ; \+ terminal(G),
       msw(first(G),Wd), lc_call(G,Wd,L,L1) ),
   g_call(R,L1,L2).

lc_call(G,B,L,L2):- % project or attach
   msw(lc(G,B),rule(A,[B|RHS2])),
   g_call(RHS2,L,L1),
   ( G == A, attach_or_project(A,Op),
       ( Op == attach, L2=L1
       ; Op == project, lc_call(G,A,L1,L2) )
   ; G \== A, lc_call(G,A,L1,L2) ).

attach_or_project(A,Op):-
   ( values(lc(A,A),_), msw(attach(A),Op)
   ; \+ values(lc(A,A),_), Op=attach ).
```

Figure 2. A probabilistic LC parser

g_call(Gs,L1,L2) says a difference list L1-L2 is derived from a list Gs of terminals and nonterminals. The shift operation in LC parsing is carried out by the second g_call clause in such a way that in the generation process, if G is not a terminal, Wd is probabilistically chosen by msw(first(G),Wd) from the first set of G. Once Wd is chosen, L-L1, the rest of sentence, is generated by calling lc_call(G,Wd,L,L1).

lc_call(G,B,L,L2) means there is a completed subtree whose category is B, the goal category G and B stand in the left-corner relation[7], and G derives [B|L]-L2 (see Figure 3). Given G and B in the generation process, a rule A $\to$ [B|RHS2] is probabilistically chosen by msw(lc(G,B),·) depending on the pair (G,B). Then g_call(RHS2,L,L1) is called to complete the derivation of L-L1 from RHS2. We have two cases then. If G = A and both project operation and attach

[7]If there is a chain of production rules $G \to A_1\alpha_1, A_1 \to A_2\alpha_2, \ldots, A_n \to B\alpha_n$ $(n \geq 1)$ G and B are said to be in the left-corner relation.

operation are possible, we probabilistically choose one of them by `msw(attach(A),·)`. Otherwise only attach operation is possible. Else $G \neq A$ and only project operation is possible. So we call `lc_call(G,A,L1,L2)`.

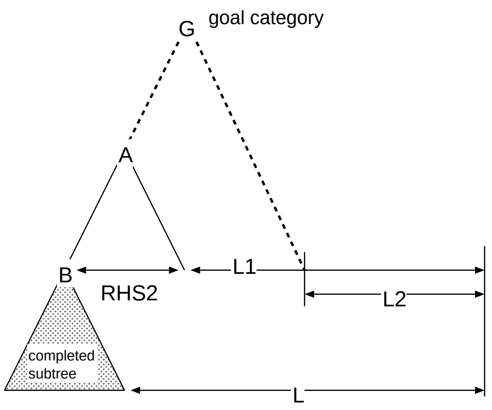

Figure 3. `lc_call(G,B,L,L2)` in LC parsing

By inspection, we know that the generation process started by `:-plc(Ws)` never fails and probabilistic choices are exclusively made by `msw/2`. Hence we may conclude according to (Sato & Kameya, 2001) that EM learning performed by the program is mathematically correct[8].

3.3. EM learning with ATR corpus

We conducted an experiment of EM learning with a real corpus, ATR corpus which is a Japanese corpus containing more than 11000 parses (Uratani et al., 1994). The backbone CFG grammar consists of 860 rules. Prior to the experiment, we specialized (unfolded) the program in Figure 2 by CFG rules for speed up. The specialization yielded about 2800 clauses such as

```
g_call(adv,[Wd|L],L2):-
   msw(first(adv),Wd),lc_call(Wd,adv,L,L2).
```

We also obtained 21000 `values` declarations for `msw` atoms. In the experiment, a PC having 3.4Ghz CPU and 2GB memory was used with a PRISM program containing these clauses and declarations. The parsing phase finished in 64 seconds and in the subsequent EM learning phase, the gEM algorithm converged after 367 iterations[9], taking 66 minutes as total learning time.

[8]Actually we need another condition "if there is no loss of probability mass to infinite generation process," which we assume to hold for simplicity here.

[9]The convergence is judged if an increase in the log-likelihood is less than 10^{-4}.

It is hard to say whether our approach is unreasonably slow or not, because (Van Uytsel et al., 2001) which is the only literature available to us on similar EM learning, does not mention the total learning time for the EM learning of a PLCG and hence comparison is difficult to make. Also we expect a new implementation of PRISM underway will speed up learning further. Putting learning time aside however, one thing seems clear: this experiment shows that our generic approach works even in the relatively unexplored field, i.e. EM learning of PLCGs, straightforwardly without much difficulty.

4. EM Learning for Probabilistic Context Free Graph Grammars

In this section, we tackle the problem of parameter learning of *PCFGGs (probabilistic context free graph grammars)*, an important yet almost unknown area. *CFGGs (context free graph grammars)* are a natural generalization of CFGs. They define a set of graphs by repeatedly replacing subgraphs using production rules whose right-hand side is a graph, not a string. PCFGGs are CFGGs with probabilities associated with production rules just like PCFGs. They define distributions over the set of producible graphs. We here focus on two classes of CFGGs and their probabilistic versions. One is *HRGs (hyper edge replacement grammars)* which replace edges and the other is *NLCGs (node label controlled grammars)* which replace nodes (Rozenberg, 1997).

As graphs are much richer structure than strings and abundantly used to describe control diagrams, chemical compounds, gene networks, WWW and so on, developing a method of parameter learning for PCFGGs would contribute greatly to the statistical analysis of complex structure represented by graphs. However in reality there is almost no literature on parameter learning of PCFGGs except (Oates et al., 2003) in which the authors state that "To the best of our knowledge, our paper is the first to present a formally sound algorithm for computing maximum likelihood parameter estimates for a large class of HR grammars."

In the following we show that PRISM enables us to learn parameters not only for *PHRGs (probabilistic HR grammars)* like (Oates et al., 2003) but for an untried class, *PNLCGs (probabilistic NLC grammars)*[10] as well.

[10]There is no literature on EM learning of PNLCGs as far as we know.

4.1. Probabilistic HR grammars

In this subsection, we attempt EM learning of PHRGs using PRISM.

4.1.1. Hyper graphs

We first introduce terminology, mostly following (Rozenberg, 1997). A *hyper graph* $\mathcal{H}$ is a triple (V, E, X) where V is a finite set of nodes, E a finite set of *hyper edges* and X a sequence of nodes in V called *external nodes*. The type of $\mathcal{H}$ is defined to be the length of X and denoted as type($\mathcal{H}$).

A *hyper edge* is an ordered pair (Cat, AN) such that Cat is a label of the hyper edge and AN is a sequence of pair-wise distinct (this is our simplifying assumption) nodes in V called *attachment nodes*. The label can be null. The length of AN is denoted by type(Cat). In Figure 4, the left graph represents a hyper edge $(S, (2, 5, 4))$ whereas the right graph represents $(S, (1, 2))$ respectively. In drawing graphs, a label is boxed and an underlined number $\underline{i}$ marks an edge connected to the i-th attachment node in AN. When the length of AN is 2, we omit underlined numbers and instead use an arrow assuming the source node of the arrow is the first attachment node and the sink node is the second one.

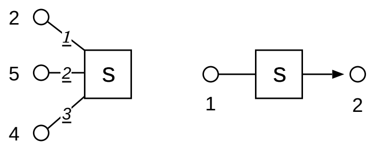

Figure 4. Hyper edges

An *HRG (hyper edge replacement grammar)* is a quadruple (N, T, P, S) where N, a set of labels called *nonterminals* and T, a set of labels called *terminals* are disjoint. P is a set of *rules* and S a *start symbol*.

A *rule* is an ordered pair (Cat, HG) where Cat is a nonterminal which is a label of a hyper edge HE whereas HG is a hyper graph such that type(Cat) = type(HG). So the attachment nodes AN in HE and the external nodes X' in HG have the same length. We understand that the i-th attachment node of AN corresponds to the i-th external node of X'. Using this correspondence, we replace (an isomorphic copy of) the hyper edge HE in a host graph $\mathcal{H}$ by HG as follows. First after matching HE against some hyper edge in $\mathcal{H}$, we remove the matched edge and add HG. Second we glue each external node of X' of HG to the corresponding attachment node of AN in $\mathcal{H}$ (*hyper edge replacement*). A *start graph* is a hyper graph such that nodes are all external nodes and it contains a single hyper edge whose label is S, the start symbol.

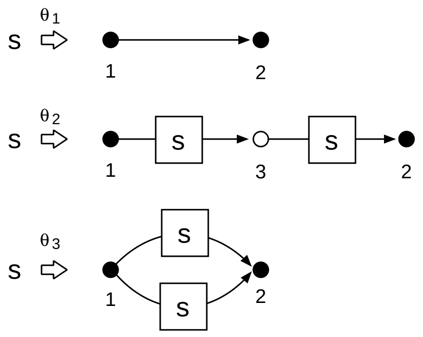

Figure 5. PHRG $\mathcal{G}_{hr}$

A *PHRG (probabilistic hyper edge replacement grammar)* is an HRG such that rules for the same label are assigned probabilities (*parameters*) which sum to unity. Just like PCFGs, we start from the start graph and repeat one step derivation, i.e. hyper edge replacement using a probabilistically chosen rule until a *terminal graph*, i.e. a hyper graph TG in which all labels are terminals, is generated. TG is a member of the set of graphs specified by the PHRG, and its probability is computed as the product of probabilities associated with rules used in its derivation. This way a PHRG defines a distribution over the set of terminal graphs. A PHRG in Figure 5 is based on an HRG borrowed from (Rozenberg, 1997). Each rule has a parameter θ_i $(i = 1, 2, 3)$. There black circles are external nodes. Figure 6 below illustrates a derivation process from the start graph in the upper-left corner.

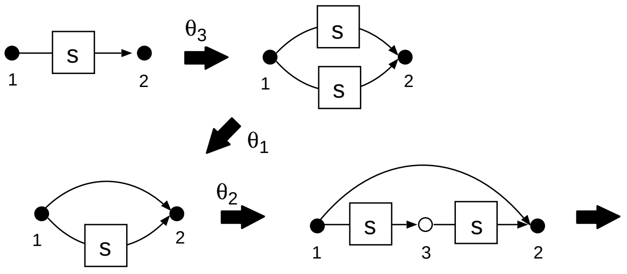

Figure 6. A derivation sequence using $\mathcal{G}_{hr}$

4.1.2. EM learning of PHRGs

We conducted EM learning of $\mathcal{G}_{hr}$. The first step is to write a PRISM program for $\mathcal{G}_{hr}$ that causes no failure, following the principle of generative modeling (Sato & Kameya, 2001). So we wrote a program DB_{hr} in Figure 7 that generatively defines a distribution over terminal graphs specified by $\mathcal{G}_{hr}$. We encoded a hyper graph as a list (set) of hyper edges represented by atoms of the form `edge(Cat,AN)`. The three rules in Figure 5 are straightforwardly encoded as a value declaration below. For example, the second line of the declaration encodes the second rule of $\mathcal{G}_{hr}$.

```
values(s,[
 [[1,2],[], [edge([],[1,2])]],
 [[1,2],[3],[edge(s,[1,3]),edge(s,[3,2])]],
 [[1,2],[], [edge(s,[1,2]),edge(s,[1,2])]] ]).

phrg(edge(Cat,AN),L):-
  get_max_node(AN,Max),
  phrg(edge(Cat,AN),Max,L,[],_).
phrg(edge(Cat,AN),Max,L1,L3,Max3):-
  ( values(Cat,_),     % replacement possible
     msw(Cat,RHS),     % choose a rule
     glue_rhs(RHS,AN,HG),
     phrg2(HG,Max2,L1,L3,Max3)
  ; \+ values(Cat,_), % no replacement
    L1 = [edge(Cat,AN)|L3],
    Max3 is Max ).
phrg2([edge(Cat,AN)|X],Max,L1,L3,Max3):-
  phrg(edge(Cat,AN),Max,L1,L2,Max2),
  phrg2(X,Max2,L2,L3,Max3).
phrg2([],Max,L1,L1,Max).
```

Figure 7. PRISM program DB_{hr} for $\mathcal{G}_{hr}$ (part)

This program works similarly to a PRISM program for a PCFG. `:-phrg(edge(s,[1,2]),L)` for example, probabilistically generates a terminal graph from `edge(s,[1,2])`, and returns it in `L`. The primary computation is done by `phrg(edge(Cat,AN),Max,L1,L3,Max3)`. When called with a ground `edge(Cat,AN)`, `L1` which stores the current hyper graph and `Max`, the current maximum number of nodes, it adds to `L1` a new terminal graph derived from `edge(Cat,AN)` and returns the enlarged terminal graph as a difference list `L1-L3` together with `Max3`, the renewed maximum node number. `glue_rhs(RHS,AN,HG)` carries out hyper edge replacement using the right-hand side of a rule and `AN`, the list of attachment nodes in the graph being constructed. It returns a terminal graph `HG` to be added to `L1`.

Using DB_{hr}, we randomly generated a sample of size 100 using parameters set to $(\theta_1, \theta_2, \theta_3) = (0.6, 0.2, 0.2)$. To cut down on processing time, the size of generated graphs was restricted to 15. To perform parameter estimation, we used a very naive parser written in PRISM which resembles DB_{hr}, and extracted explanation graphs from the sampled graphs, on which the gEM algorithm was run to estimate parameters[11]. We repeated this experiment 20 times and took averages of estimated parameters. The table in Figure 8 summarizes results where σ is a standard deviation. Looking at these figures, it might be safely said that parameters are reasonably estimated though how the size restriction on generated graphs affected estimation is unclear.

	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
Original	0.6	0.2	0.2
Ave.	0.65082	0.18220	0.16697
σ	0.01949	0.01672	0.01692

Figure 8. Learned parameters for $\mathcal{G}_{hr}$

We conducted EM learning of PHRGs for other probabilistic models as well. They include an HMM, a PCFG, a probabilistic context sensitive grammar and so on, and in all cases we could successfully estimate parameters.

4.2. Probabilistic NLC grammars

Finally we attempt EM learning of probabilistic *NLC (node label controlled) grammars*. Due to the lack of space, we have to skip details and programs.

Suppose we have NT, a set of nonterminal node labels, and T, a set of terminal node labels. NT and T are disjoint. As convention we use upper case letters for labels in NT and lower case letters for those in T. Next an *NLC rule* is a replacement rule $X \rightarrow R : C$ where X is a nonterminal node label, R is an undirected graph called *replacing graph* such that nodes are labeled by $NT \cup T$. In the following, we add a simplifying assumption that *labels in R are distinct from one another*. So R can be represented by a set of pairs of labels. C denotes a set of connection instructions (α, β) where α and β are from $NT \cup T$. Let us take an example to get the idea of node replacement using C.

$$A \rightarrow \{(a, A)\} : \{(a, a), (a, A)\}$$

This rule replaces a node n labeled A in a host graph H with a new graph in three steps. First we remove n and every edge incidental to n from H. We use H^- to refer to the remaining graph. Second we add (a copy of) the replacing graph $R = \{(a, A)\}$ to H^-. Here (a, A) is an edge connecting a node labeled a and

[11]We did not attempt to develop a sophisticated parser, as our primary purpose is not efficient parsing but the verification of the possibility of EM learning.

a node labeled A. Finally according to a connection instruction (a,a) in the right component of the rule, we connect every node labeled a in H^- to the node labeled a in the added R. Likewise we process (a,A) by connecting every node labeled a in H^- to every node labeled A in R.

An *NLC grammar* is a tuple (NT, T, P, S) where NT is a set of nonterminal node labels, T a set of terminal node labels, P a set of NLC rules and S a start graph (we assume a single symbol here). By associating probabilities with rules as usual, we can define a *PNLCG (probabilistic NLC grammar)*. Figure 9 is an example of probabilistic NLC grammar $\mathcal{G}_{nlc}$.

$$\left\{\begin{array}{lllllll}
A & \to & \{(a,A)\} & : & \{(a,a),(a,A)\} & : & \theta_1 \\
A & \to & \{(a,B)\} & : & \{(a,a),(a,B)\} & : & \theta_2 \\
A & \to & \{a\} & : & \{(a,a)\} & : & \theta_3 \\
B & \to & \{(a,A)\} & : & \{(a,a),(a,A)\} & : & \theta_4 \\
B & \to & \{(a,B)\} & : & \{(a,a),(a,B)\} & : & \theta_5 \\
B & \to & \{a\} & : & \{(a,a)\} & : & \theta_6
\end{array}\right.$$

Figure 9. $\mathcal{G}_{nlc}$

Figure 10 depicts a derivation of a terminal graph which has no nonterminals. Nonterminal labels are boxed while terminal labels are put in a circle. In this case the probability of the derived graph is calculated as $\theta_1\theta_2\theta_4\theta_3$.

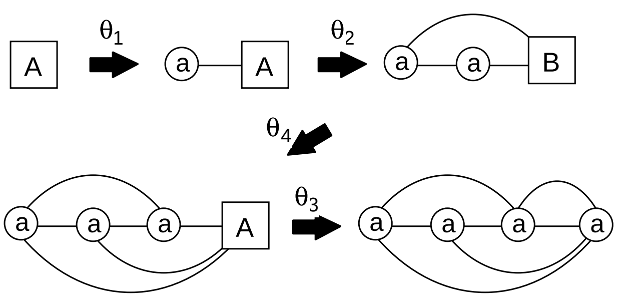

Figure 10. A derivation sequence by $\mathcal{G}_{nlc}$

We conducted EM learninng of $\mathcal{G}_{nlc}$. We first wrote a PRISM program DB_{nlc} not shown here which faithfully simulates a derivation process of $\mathcal{G}_{nlc}$. After setting parameters to $(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = (0.2, 0.2, 0.6, 0.6, 0.2, 0.2)$, we randomly generated 1000 samples by DB_{nlc} with A being a start symbol. We also prepared a simple (naive) parser written in PRISM by slightly modifying DB_{nlc} and let it run on the sampled graphs to learn parameters by the gEM algorithm. the gEM algorithm stopped after 63 iterations (threshold is 10^{-4}). The estimated parameters are shown in Table 11. They appear to be close to the

	$\hat{\theta}_1$	$\hat{\theta}_2$	$\hat{\theta}_3$
Original	0.2	0.2	0.6
Estimate	0.22134	0.21907	0.5595

	$\hat{\theta}_4$	$\hat{\theta}_5$	$\hat{\theta}_6$
Original	0.6	0.2	0.2
Estimate	0.60468	0.15049	0.24481

Figure 11. Learned parameters for $\mathcal{G}_{hr}$

original values[12].

5. Discussion and related work

We have proposed a generic approach to EM learning for symbolic-statistical models based on a logic-based probabilistic language PRISM (Sato & Kameya, 1997). It saves us the trouble of deriving a specialized EM algorithm for each application. We have only to write, obeying the principle of generative modeling (Sato & Kameya, 2001), a PRISM program tuned for each probabilistic model to define a desired distribution. Then parameters are automatically estimated from data by a built-in EM algorithm.

We have demonstrated effectiveness of our approach by tackling problems in relatively unknown areas, i.e. EM learning for a probabilistic LC parser, one for a probabilistic HR graph grammar and one for a probabilistic NLC graph grammar, where very few or no literature is available. We developed appropriate PRISM programs and EM learning was successfully done in every case using them, which seems to imply that PRISM is an appropriate tool for rapid prototyping of new probabilistic models.

Recently there are many proposals for probabilistic language for probabilistic models. IBAL (Pfeffer, 2001) for example is a probabilistic functional language which can define distributions and perform parameter estimation. It also can deal with decision theoretic programs. Dyna (Eisner et al., 2004) is a programming language designed mainly for statistical natural language processing. It is based on equations and dynamic programming. For other proposals related to statistical relational learning, (De Raedt & Kersting, 2003) provides a comprehensive survey.

References

De Raedt, L., & Kersting, K. (2003). Probabilistic logic learning. *ACM-SIGKDD Explorations, special*

[12]We could not repeat this experiment due to a shortage of time.

issue on Multi-Relational Data Mining, *5*, 31–48.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Royal Statistical Society*, *B39*, 1–38.

Doets, K. (1994). *From logic to logic programming.* The MIT Press.

Eisner, J., Goldlust, E., & Smith, N. (2004). Dyna: A declarative language for implementing dynamic programs. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL'04), Companion Volume* (pp. 218–221).

Elston, R., & Stewart, J. (1971). A general model for the genetic analysis of pedigree data. *Human Heredity*, *21*, 523–542.

Inui, K., Sornlertlamvanich, V., Tanaka, H., & Tokunaga, T. (1997). *A new probabilistic LR language model for statistical parsing*Technical Report TR97-0004 Dept. of CS). Tokyo Institute of Technology.

Kameya, Y., & Sato, T. (2000). Efficient EM learning for parameterized logic programs. *Proceedings of the 1st Conference on Computational Logic (CL2000)* (pp. 269–294). Springer.

Kruglyak, L., Daly, M., Reeve-Daly, M., & Lander, E. (1996). Parametric and nonparametric linkage analysis: A unified multipoint approach. *American Journal of Human Genetics*, *58*, 1347–1363.

Manning, C. (1997). Probabilistic parsing using left corner language models. *Proceedings of the Fifth International Conference on Parsing Technologies (IWPT-97)* (pp. 147–158). MIT Press.

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing.* The MIT Press.

McLachlan, G. J., & Krishnan, T. (1997). *The EM algorithm and extensions.* Wiley Interscience.

Oates, T., Doshi, S., & Huang, F. (2003). Estimating maximum likelihood parameters for stochastic context-free graph grammars. *Proceedigs of the 13th International Conference on Inductive Logic Programming (ILP 2003)* (pp. 281–298).

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems.* Morgan Kaufmann.

Pfeffer, A. (2001). IBAL: A probabilistic rational programming language. *Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI'01)* (pp. 733–740).

Rabiner, L. R., & Juang, B. (1993). *Foundations of speech recognition.* Prentice-Hall.

Roark, B., & Johnson, M. (1999). Efficient probabilistic top-down and left-corner parsing. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (pp. 421–428).

Rozenberg, G. (Ed.). (1997). *Handbook of graph grammars and computing by graph transformations, volume 1: Foundations.* World Scientific.

Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)* (pp. 715–729).

Sato, T., & Kameya, Y. (1997). PRISM: a language for symbolic-statistical modeling. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)* (pp. 1330–1335).

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, *15*, 391–454.

Sato, T., & Kameya, Y. (2004). A dynamic programming approach to parameter learning of generative models with failure. *Proceedings of ICML 2004 workshop on Learning Statistical Models from Relational Data (SRL2004).*

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

Van Uytsel, D., Van Compernolle, D., & Wambacq, P. (2001). Maximum-likelihood training of the PLCG-based language model. *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop 2001 (ASRU'01).*

Zhou, N.-F., Shen, Y., & Sato, T. (2004). Semi-naive Evaluation in Linear Tabling . *Proceedings of the Sixth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP2004)* (pp. 90–97).