Yet more efficient EM learning for parameterized logic programs by inter-goal sharing

Yoshitaka Kameya[1] and **Taisuke Sato**[2] and **Neng-Fa Zhou**[3]

Abstract. In previous research, we presented a dynamic-programming-based EM (expectation-maximization) algorithm for parameterized logic programs, which is based on the structure sharing with tabled search. It is also shown that this general framework achieves the same time complexity as that of the specialized algorithms. The efficiency is brought by sharing common (partial) paths in the derivation tree for a given goal, but such sharing is incomplete in the sense that it is not allowed to share the paths appearing in different derivation trees. In this paper, we introduce a general idea called 'inter-goal sharing' where different goals can share the common derivation paths. Inter-goal sharing achieves the full sharing of derivation paths and hence makes EM learning more compact and efficient in practical cases. Experimental results with both artificial and real linguistic data show that the proposed method runs 2-6 times more compactly and faster than the previous approach.

1 Introduction

To build an AI system that deals with uncertainty, one promising way is to integrate domain knowledge seamlessly with well-founded statistical techniques. Since it seems not easy to describe such domain knowledge only with propositional languages, researches on the integration between first-order logic and probability [2, 4, 8, 10, 13, 15] have been paid much attention in the last decade. In particular, based on the distribution semantics, a probabilistic extension of least model semantics of logic programs, we proposed parameterized logic programs, called PRISM (programming in statistical modeling) programs [15], each of which represents a probability distribution parameterized with basic probabilities (called *parameters*) of choices. In PRISM, we model a sequential process of probabilistic choices, which leads to our observation. Inversely, from a collection of observations (i.e. training data), we can find the maximum likelihood (ML) estimate of such parameters by considering all sequences of choices for each observation, and by applying the EM (expectation-maximization) algorithm [5], as is done for hidden Markov models (HMMs) or probabilistic context-free grammars (PCFGs) [15]. Each observation is here called a (top) *goal*, and a sequence of choices leading to a goal can be seen as an *explanation* path for the goal.

In recent work, we proposed an efficient framework for EM learning of PRISM programs, which combines tabled search (memoization) technique [19, 22, 23] for logic programs and a dynamic-programming-based EM algorithm called *the graphical EM* (gEM) algorithm. The efficiency is brought by sharing common (partial) paths in the derivation tree for a given goal, and by designing the gEM algorithm to run on this compact shared structure. However, it can be said that such sharing is incomplete in the sense that it is not allowed to share the paths appearing in the derivation trees for different goals. So the redundancies will grow as we attempt to collect a larger set of observations to ensure the statistical reliability of estimated parameters. A motivative example is given in Section 2.

In this paper, we present a general idea called *inter-goal sharing* where different goals can share common derivation paths. Inter-goal sharing achieves the full sharing of derivation paths and hence makes EM learning more compact and efficient in practical cases. In addition, we give a simple implementation of inter-goal sharing for PRISM programs, which can be justified in both logical and probabilistic senses. Finally we show the experimental results with two typical and widely-applied statistical language models, i.e. HMMs and PCFGs. For both artificial and real linguistic data it is found that the proposed EM algorithm runs 2-6 times more compactly and faster than the previous approach.

2 Background

2.1 PRISM programs

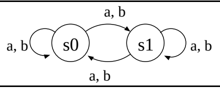
We start by explaining PRISM language with a small program example of an hidden Markov model (HMM) [3, 12]. A detailed description of PRISM is given in [15]. A PRISM program shown in Figure 1 represents an HMM which has two states $\{\texttt{s0},\texttt{s1}\}$ and outputs a symbol '`a`' or '`b`' at each state. To define a probability distribution, PRISM provides a built-in predicate `msw`/2.[4] A ground atom `msw(`i`,` v`)` means that a multi-outcome switch i takes a value v with the probability $\theta_{i,v}$, by which we can represent a probabilistic choice identified by i which results in v. For each i, the second argument v of `msw`/2 comes from a set V_i of ground terms, and $\sum_{v\in V_i}\theta_{i,v}=1$ holds. Each V_i is given by a built-in predicate `value`/2 as shown in Figure 1 (e.g. $V_{\texttt{tr(}\cdot\texttt{)}}=\{\texttt{s0},\texttt{s1}\}$). In the rest of the paper, a 'switch' refers to a ground atom of `msw`/2. Also we assume that all trials of `msw`/2 are independent, following the current version of PRISM implementation [24]. In Figure 1, a procedural reading of the HMM program as a string generator is given as comments.

2.2 Efficient EM learning with tabling

The EM algorithm is a well-known class of numerical algorithms for ML estimation of a probability model with a hidden structure. In the case of PRISM programs, explanation paths are hidden from the observations (i.e. top goals). This situation requires an EM algorithm to get the ML estimation of the parameters $\theta_{i,v}$ of switches.

[1] Tokyo Institute of Technology, kameya@mi.cs.titech.ac.jp
[2] Tokyo Institute of Technology and CREST, sato@mi.cs.titech.ac.jp
[3] The City University of New York, zhou@sci.brooklyn.cuny.edu

[4] In this paper, we use the notation p/n for a predicate p with arity n.



```
values(init,[s0,s1]).  % for initial state dist.
values(out(_),[a,b]).  % for output dist.
values(tr(_),[s0,s1]). % for transition dist.

hmm(L):-             % To generate a string L,
   msw(init,S),      %   Set initial state to S
   hmm(1,S,L).       %   Enter the loop (clock=1)

hmm(T,S,[C|L]):- T=<3, % Loop:
   msw(out(S),C),      %   Output C at state S
   msw(tr(S),Next),    %   Transit S -> Next
   T1 is T+1,          %   Put the clock ahead
   hmm(T1,Next,L).     %   Continue the loop
hmm(T,_,[]):- T>3.     % Finish the loop
```

Figure 1. A two-state HMM (above) and its PRISM program (below).

Specifically for HMMs, an efficient EM algorithm is known as the forward-backward or the Baum-Welch algorithm [3, 12]. As easily seen from Figure 1, PRISM programs are a generalization of HMMs in expressivity, so we need an EM algorithm for PRISM programs which is general but still runs as fast as specialized EM algorithms (e.g. the Baum-Welch algorithm). Previous papers (e.g. [15]) show that a framework with a combination of tabled search and the graphical EM (gEM) algorithm fulfills this requirement.[5] Roughly speaking, in tabled search, a subgoal is tabled (memorized) with the answer when it is proved, and the tabled subgoals behave as already known facts when they are called again. Hence the derivation paths for different calls of a subgoal will be shared. We use a data structure, called *explanation graphs*, which intermediates between tabled search and the gEM algorithm. That is, given a PRISM program and a multi-set of observed goals $\mathcal{G} = \langle G_1, \ldots, G_T \rangle$, an explanation graph for G_t

```
hmm([a,b,a])
  → { hmm(1,s0,[a,b,a]), msw(init,s0) }
  → { hmm(1,s1,[a,b,a]), msw(init,s1) }

hmm(1,s0,[a,b,a])
  → { hmm(2,s0,[b,a])†, msw(out(s0),a), msw(tr(s0),s0) }
  → { hmm(2,s1,[b,a]), msw(out(s0),a), msw(tr(s0),s1) }

hmm(1,s1,[a,b,a])
  → { hmm(2,s0,[b,a])†, msw(out(s1),a), msw(tr(s1),s0) }
  → { hmm(2,s1,[b,a]), msw(out(s1),a), msw(tr(s1),s1) }

hmm(2,s0,[b,a])‡
  → { hmm(3,s0,[a]), msw(out(s0),b), msw(tr(s0),s0) }
  → { hmm(3,s1,[a]), msw(out(s0),b), msw(tr(s0),s1) }

hmm(2,s1,[b,a])
  → { hmm(3,s0,[a]), msw(out(s1),b), msw(tr(s1),s0) }
  → { hmm(3,s1,[a]), msw(out(s1),b), msw(tr(s1),s1) }

hmm(3,s0,[a])
  → { msw(out(s0),a), msw(tr(s0),s0) }
  → { msw(out(s0),a), msw(tr(s0),s1) }

hmm(3,s1,[a])
  → { msw(out(s1),a), msw(tr(s1),s0) }
  → { msw(out(s1),a), msw(tr(s1),s1) }
```

Figure 2. An explanation graph for a single goal `hmm([a,b,a])`.

[5] Furthermore, more specifically to PCFGs, we can use a sophisticated CFG parsers instead of the interpreters of logic programming systems. It is reported in [14] that, with a moderately ambiguous CFG skeleton [20, 21], the gEM algorithm runs faster than the Inside-Outside algorithm (a specialized EM algorithm for PCFGs) by orders of magnitude.

is extracted from the table used in the search of explanations for G_t ($1 \leq t \leq T$). Then the gEM algorithm runs on these explanation graphs to estimate $\theta_{i,v}$.

Using the terminology in [24], explanation graphs are formulated as follows. A *explanation path* for a tabled subgoal τ is defined as ($\tau \to S$) where S is a set of tabled subgoals or switches. An explanation path corresponds to an instance of a clause where all tabled subgoals or switches in S appear in the body of the clause. τ is called the *root* of the path. An *explanation tree* $\psi(\tau)$ for a tabled subgoal τ is a set of possible explanation paths for τ (τ is also called the root of the tree). An *explanation graph* for a goal G_t ($1 \leq t \leq T$) is now defined as a pair $\langle \mathcal{O}_t, \psi \rangle$ where $\mathcal{O}_t$ is an ordered set $\langle \tau_1, \ldots, \tau_{K_t} \rangle$ of distinct roots, and $\mathcal{O}_t$ should be acyclic, i.e. it should hold that $G_t = \tau_1$ and that if $k \leq k'$ and ($\tau_{k'} \to S$) is an explanation path for $\tau_{k'}$ then τ_k does not appear in S. This acyclicity among roots is guaranteed by topological sorting based on the calling relationship.

Figure 2 illustrates an explanation graph for a single goal `hmm([a,b,a])` in the HMM program. It can be found that structure sharing is made, by seeing two tabled subgoals labeled by '†' refer to an explanation tree rooted by the one with '‡'. Also, the guarantee of the acyclicity allows us to make a dynamic programming approach to EM learning. From the description of the gEM algorithm given in [15], it is easy to find that the algorithm runs in linear time of $\xi = \sum_{t=1}^{T} \sum_{\tau \in \mathcal{O}_t} \sum_{S:(\tau \to S) \in \psi(\tau)} |S|$,[6] the total size of explanation graphs. Notice here that the gEM algorithm runs slower as T becomes larger (i.e. the training data grows). This can arise a problem when we prepare a large data to ensure the statistical reliability of estimated parameters. Also it may be more severe that the size of required memory is also $O(\xi)$, so there would be cases where the gEM algorithm does not work on the computers at hand.

3 Inter-goal sharing

The purpose of this paper is to improve the capacity of tabled search and the gEM algorithm by preventing the growth of explanation graphs in practical cases. For such improvement, we introduce the idea of *inter-goal sharing*, which attempts to share the subgoals among top goals G_t, whereas in previous work, the structure sharing is separately made for each top goal (i.e. intra-goal sharing). For instance, let us see Figure 2 and consider another top goal `hmm([b,b,a])`. Then it can be easily seen that the subgoal `hmm(2,s0,[b,a])` occurs in the derivation path of both top goals `hmm([a,b,a])` and `hmm([b,b,a])`, and can be shared.

Fortunately for PRISM programs, it is quite easy to derive an algorithm which implements inter-goal sharing. First of all, suppose that a PRISM program and observed goals $\mathcal{G} = \langle G_1, G_2, \ldots, G_T \rangle$ is given and let us rename the logical variables in G_t so that there is no variable appearing in more than one goal from $\mathcal{G}$. Then, with a new predicate `pseudo_goal`/0, inter-goal sharing will be done by only adding a clause

$$\texttt{pseudo\_goal:-}\ G_1, G_2, \ldots, G_T\,. \qquad (*)$$

to the original program and regarding `pseudo_goal` as a single observed goal. In tabled search, if the subgoals appearing in the search for G_t is shared by any succeeding sibling $G_{t'}$ ($t < t'$), we can say inter-goal sharing is made between G_t and $G_{t'}$. Also, it is easy to show the topological sorting from `pseudo_goal` naturally resolves $\mathcal{O}_t$ ($1 \leq t \leq T$) into a single (large) ordered set $\mathcal{O}$ of distinct roots (notice that ψ can be considered as being common to t).

[6] For HMM programs like Figure 1, $\xi = O(N^2 LT)$, where N is the number of states, and L is the length of an input sequence.

In the context of EM learning (ML estimation), on the other hand, adding the dummy clause $(*)$ is justified based on the distribution semantics [15]. Firstly, in a PRISM program, we have assumed that all trials of msw/2 are independent (see Section 2.1). From this and renaming of variables in $\mathcal{G}$, it is obvious that the goals from $\mathcal{G}$ are all independent. Besides, since $(*)$ is a unique clause for pseudo_goal/0, we can think of the head pseudo_goal as logically equivalent to a conjunction of G_t, and hence $P(\texttt{pseudo\_goal}) = \prod_{t=1}^{T} P(G_t)$ (from the independence assumption among goals). Now it can be concluded that maximizing $P(\texttt{pseudo\_goal})$, the likelihood of pseudo_goal in the modified program, is equivalent to maximizing $\prod_{t=1}^{T} P(G_t)$, the likelihood of $\mathcal{G}$ in the original program.

4 Modified EM algorithm

Since the modification of PRISM programs described in the previous section does not change in both logical and probabilistic senses, we don't need to modify the original gEM algorithm [15]. However, there still remain a redundancy if some goal appears repeatedly in $\mathcal{G}$. To avoid this, as is usually done, let us consider $\{G^{(1)}, \ldots, G^{(M)}\}$ as a set of distinct goals in $\mathcal{G}$, and introduce *weights* w_m as the numbers of occurrences of $G^{(m)}$ in $\mathcal{G}$ $(1 \leq m \leq M)$. Then the redundancy in $\mathcal{G}$ will be removed by recording weights and redefining the clause for pseudo_goal/0 as follows:

$$\texttt{pseudo\_goal:-}\ G^{(1)}, G^{(2)}, \ldots, G^{(M)}.$$

Figure 3 shows a modified gEM algorithm taking weights into account. The central part of the algorithm is the computation of $\eta[i, v]$, the expected occurrences of $\texttt{msw}(i,v)$, by which the parameters $\theta_{i,v}$ are repeatedly updated. Compared to the original algorithm, there are only two places that need to be modified. First, we modify *learn-gEM* and *GetInsideProbs* so that they run on a single (large) explanation graph. Secondly, *GetExpectations* is modified so that, only for each top goal $G^{(m)}$, the variable $\mathcal{Q}[G^{(m)}]$ is initialized to $w_m \mathcal{P}[\texttt{pseudo\_goal}]/\mathcal{P}[G^{(m)}]$ (the line (b)). Considering the explanation graph obtained by the dummy clause $(*)$, and noting that $\mathcal{P}[\texttt{pseudo\_goal}] = \prod_{t=1}^{T} \mathcal{P}[G_t]$, it can be easily shown that this initial value is just the sum of outside probabilities of goals in $\mathcal{G}$ each of which is identical to $G^{(m)}$, and hence the modified algorithm yields the same result as that of the original algorithm.[7] Also we can see that, in the modified version, the weights are smoothly accumulated to $\eta[i, v]$ along with the dynamic-programming-based computation.

5 Experiments

5.1 Hidden Markov models

To show the efficiency of the proposed algorithm, we have conducted experiments with two widely-applied language models, i.e. HMMs and PCFGs.[8] The HMM program in Figure 1 is used for the experiment. First we sampled sufficiently many goals by PRISM system's built-in,[9] and fixed them as $\mathcal{G}_0$. Then, as a training data for the HMM program, we repeatedly pick up the first T goals from $\mathcal{G}_0$ with increasing T by 100. Since the modified gEM algorithm runs in linear time of the size ξ of the explanation graph, we only measured ξ after tabled search, instead of the real running time.[10] For the length L of input strings, we tried two cases: $L = 10$ and $L = 15$.[11] Also three types of gEM algorithms are tested: the original one in [15] (referred to as "*Original*"), the modified one in Figure 3 ("*Modified*"), and the one which does not make inter-goal sharing but only removes the redundancy in training data by taking weights into account ("*Weights*"). Figure 4 shows the results. When $L = 10$, inter-goal sharing is quite effective, i.e. the modified algorithm runs 6 times more compactly and faster than the original one (at $T = 1,000$). The algorithm only taking weights into account also runs fast because the number of distinct observable patterns are $2^{10} = 1,024$ (we have only two output symbols and $L = 10$). Obviously, this is not the case when $L = 15$, whereas the modified algorithm still runs 3 times faster than the original (at $T = 3,000$). For longer sequences, unfortunately, the effectiveness of compaction will be decreased since

```
procedure Learn-gEM($\mathcal{O}$, $\psi$) begin
  Select some parameters $\theta$ and let $\tau_0$ = pseudo_goal;
  GetInsideProbs($\mathcal{O}$, $\psi$);
  $\lambda^{(0)} := \log \mathcal{P}[\tau_0]$;
  repeat
    GetExpectations($\mathcal{O}$, $\psi$);
    foreach $i \in I, v \in V_i$ do $\eta[i,v] := \eta[i,v]/\mathcal{P}[\tau_0]$;          $\cdots$ (a)
    foreach $i \in I, v \in V_i$ do $\theta_{i,v} := \eta[i,v]/\sum_{v' \in V_i} \eta[i,v']$;
    GetInsideProbs($\mathcal{O}$, $\psi$);
    $\lambda^{(m)} := \log \mathcal{P}[\tau_0]$;
    $m := m + 1$
  until $\lambda^{(m)} - \lambda^{(m-1)} < \varepsilon$
end

procedure GetInsideProbs($\mathcal{O}$, $\psi$) begin
  Let $\mathcal{O} = \langle$pseudo_goal$, \tau_1, \ldots, \tau_K\rangle$ and $\tau_0$ = pseudo_goal;
  for $k := K$ downto 0 do begin
    $\mathcal{P}[\tau_k] := 0$;
    foreach $S$ such that $(\tau_k \to S) \in \psi(\tau_k)$ do begin
      $\mathcal{R}[\tau_k, S] := 1$;
      foreach $A \in S$ do
        if $A$ = msw($i,v$) then $\mathcal{R}[\tau_k, S]$ *= $\theta_{i,v}$
        else $\mathcal{R}[\tau_k, S]$ *= $\mathcal{P}[A]$;
      $\mathcal{P}[\tau_k]$ += $\mathcal{R}[\tau_k, S]$
    end
  end
end

procedure GetExpectations($\mathcal{O}$, $\psi$) begin
  Let $\mathcal{O} = \langle$pseudo_goal$, \tau_1, \ldots, \tau_K\rangle$ and $\tau_0$ = pseudo_goal;
  foreach $i \in I, v \in V_i$ do $\eta[i,v] := 0$;
  for $k := 1$ to $K$ do
    if $\exists m\ (\tau_k = G^{(m)})$ then $\mathcal{Q}[\tau_k] := w_m \mathcal{P}[\tau_0]/\mathcal{P}[G^{(m)}]$     $\cdots$ (b)
    else $\mathcal{Q}[\tau_k] := 0$;
  for $k := 1$ to $K$ do
    foreach $S$ such that $(\tau_k \to S) \in \psi(\tau_k)$ do
      foreach $A \in S$ do
        if $A$ = msw($i,v$) then $\eta[i,v]$ += $\mathcal{Q}[\tau_k] \cdot \mathcal{R}[\tau_k, S]$
        else $\mathcal{Q}[A]$ += $\mathcal{Q}[\tau_k] \cdot \mathcal{R}[\tau_k, S]/\mathcal{P}[A]$
end
```

Figure 3. Modified gEM algorithm.

[7] For the further optimization in Figure 3, some might notice that the occurrences of the likelihood $\mathcal{P}[\tau_0]$ in lines (a) and (b) can be canceled. That is, it is allowed that we simultaneously delete the line (a) and replace the initial value $w_m \mathcal{P}[\tau_0]/\mathcal{P}[G^{(m)}]$ by $w_m/\mathcal{P}[G^{(m)}]$ in line (b).

[8] As for the memory problem, version 1.7 of PRISM (http://sato-www.cs.titech.ac.jp/prism/), which is build on B-Prolog (http://www.probp.com/), already adopts inter-goal sharing.

[9] In sampling, we used the following parameters $\theta_{i,v}$:

i	v	
	s0	s1
init	0.9	0.1

i	v	
	a	b
out(s0)	0.5	0.5
out(s1)	0.6	0.4

i	v	
	s0	s1
tr(s0)	0.2	0.8
tr(s1)	0.8	0.2

[10] There is also an implementational reason. Currently we have implemented the algorithm in Figure 3 only as a Prolog code, which is quite slow.

[11] Figure 1 is the program with $L = 3$.

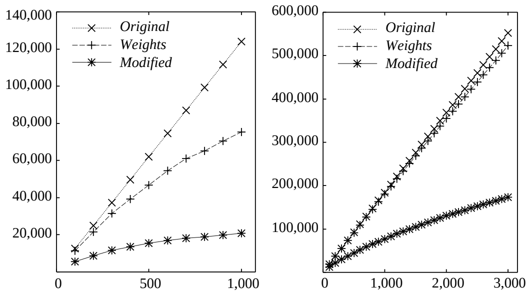


Figure 4. Experimental results for HMM: $L = 10$ (left), $L = 15$ (right), where x-axis and y-axis correspond to the size T of training data and the size ξ of the explanation graph(s), respectively.

there is less chance for each subgoal to be shared. On the other hand, for the fully connected HMM, the effectiveness does not change, because for each time t and subsequence w, subgoals `hmm(`t`,`s`,`w`)` occur in the explanation graphs symmetrically with respect to the state s (see Figure 2).

5.2 Probabilistic context-free grammars

PCFGs are a probabilistic extension of context-free grammars. Each production rule $A \to \alpha$ has a probability $\theta_{A\to\alpha}$, where A is a nonterminal symbol and α is a sequence of terminal or non-terminal symbols, and for every non-terminal A, $\sum_\alpha \theta_{A\to\alpha} = 1$ should hold.[12] Figure 5 shows Charniak's example of a PCFG [3]. For the experiment, we wrote two types of PRISM programs which represent PCFGs. One is a probabilistic version of DCG (definite clause grammar) programs [18] described in [16], and the other is a program using an auxiliary non-probabilistic predicate `divide`/3 (referred to as "PCFG-div"). Following [16], we describe the Charniak's example in Figure 6 (above) as a probabilistic DCG (PDCG) program. A probabilistic event that a rule $A \to \alpha$ is chosen at nonterminal A is represented by a switch `msw(`A`,`V_α`)` (V_α is a Prolog list of symbols appearing in α). The predicate `pdcg2(`A`,`W`)` means that a substring W is governed by a nonterminal A. In tabled subgoals obtained after tabled search, the second argument W is instantiated by a (ground) difference list `[`w_k`,...,`w_L`]-[`$w_{k'+1}$`,...,`w_L`]`, which indicates a substring $w_k w_{k+1} \cdots w_{k'}$.[13] On the other hand, Figure 6 (below) shows a PCFG-div program for the Charniak's example. Just like `pdcg2`/2 in the PDCG program, a tabled subgoals `pcfg_div2(`A`,`W`)` also says that a substring W is governed by a nonterminal A, but syntactically W is just a list `[`w_k`,...,`$w_{k'}$`]` meaning $w_k w_{k+1} \cdots w_{k'}$. The predicate `divide`/3 is defined as non-deterministic so that we can find all boundaries of phrases.[14] In a probabilistic sense, we are able to translate PDCG programs and PCFG-div programs interchangeably, i.e., by considering proof trees by (non-tabled) exhaustive search, it is possible to make one-to-one correspondences between successful paths for a PDCG program and those for the translated PCFG-div program, and vise versa.

s	$\to$	np vp	(0.8)	vp	$\to$	verb	(0.3)	verb	$\to$	swat	(0.2)
s	$\to$	vp	(0.2)	vp	$\to$	verb np	(0.3)	verb	$\to$	flies	(0.4)
np	$\to$	noun	(0.4)	vp	$\to$	verb pp	(0.2)	verb	$\to$	like	(0.4)
np	$\to$	noun pp	(0.4)	vp	$\to$	verb np pp	(0.2)	noun	$\to$	swat	(0.05)
np	$\to$	noun np	(0.2)	prep	$\to$	like	(1.0)	noun	$\to$	flies	(0.45)
pp	$\to$	perp np	(1.0)					noun	$\to$	ants	(0.5)

Figure 5. Charniak's example of a PCFG ('s' is the start symbol).

```
values(s, [[np,vp],vp]).
values(np, [noun,[noun,pp],[noun,pp]]).
  :
pdcg(L):- pdcg2(s,L-[]).

pdcg2(np,[F|L0]-L1):- first(np,F),
    msw(np,noun),
    pdcg2(noun,[F|L0]-L1).

pdcg2(np,[F|L0]-L1):- first(np,F),
    msw(np,[noun,pp]),
    pdcg2(noun,[F|L0]-X0),pdcg2(pp,X0-L1).
  :
```

```
pcfg_div(L):- pcfg_div2(s,L).

pcfg_div2(np,[F|L]):- first(np,F),
    msw(np,noun),
    pcfg_div2(noun,[F|L]).

pcfg_div2(np,[F|L]):- first(np,F),
    msw(np,[noun,pp]),
    divide([F|L],L0,L1),
    pcfg_div2(noun,L0),pcfg_div2(pp,L1).
  :
divide([A,B|As],[A],[B|As]).
divide([A|As],[A|Bs],Cs):- divide(As,Bs,Cs).
```

Figure 6. A PDCG program (above) and a PCFG-div program (below).

For these two types of PCFG programs, we made experiments with artificial and real data. For an artificial data, we sampled sufficiently many goals by using both the CFG skeleton and rule probabilities in Figure 5. On the other hand, the ATR corpus [21] and a manually developed Japanese CFG [20] are used as a real data.[15] Then, as we did in the HMM case, we measure the size ξ of the explanation graph with the size T of training data being increased by 100. Figure 7 shows the results. For the artificial data with the Charniak's example, it can be observed that the modified algorithm runs about 3–4 times faster than the original, and that the PCFG-div program runs more efficiently than the PDCG program. The difference in performance between PDCG and PCFG-div comes from the substring representation (the second argument W of `pdcg2`/2) in tabled subgoals. That is, a difference list contains the information of a suffix substring, which make the subgoals have less chance to be shared (in both inter-goal and intra-goal). Also, the difference of performance caused by the substring representation turns to be more significant in the case of the ATR corpus. In addition, from the result for the ATR corpus, we can find that the modified EM algorithm achieved to run twice faster than the original one.[16]

[12] For simplicity, we assume here that the grammar has no ε-rule, and for any nonterminal A, there is no derivation from A to A itself.

[13] In addition, attempting to reduce the burden of tabled search, we compute in advance the FIRST set for each nonterminal. `first(`A`,`x`)` means that x is an element of the FIRST set of A. Now we are concentrating on EM learning (and hence on exhaustive tabled search), both the PDCG and PCFG-div program are specialized to parsing.

[14] More concretely, a query "`?- divide([w1,w2,w3,w4],L1,L2).`" yields three answers: {`L1=[w1],L2=[w2,w3,w4]`}, {`L1=[w1,w2],L2=[w3,w4]`}, {`L1=[w1,w2,w3],L2=[w4]`}.

[15] The numbers of rules, nonterminal symbols, and terminals (part-of-speeches) are 860, 173, and 441, respectively. The sentences in the ATR corpus were randomly re-ordered in advance.

[16] It was observed that, in tabled search, the PDCG program runs faster than the PCFG-div program. It is possible that `divide`/3 in the PCFG-div program blindly determines the phrase boundaries, and hence tends to produce

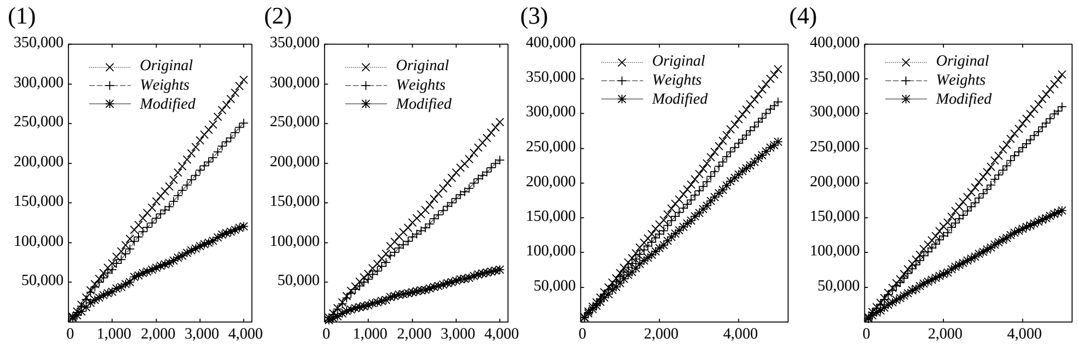


Figure 7. Experimental results for Charniak's example with PDCG (1) and PCFG-div (2), and for ATR corpus with PDCG (3) and PCFG-div (4), where x-axis and y-axis correspond to the size T of training data and the size ξ of the explanation graph(s), respectively.

6 Related work and Conclusions

In this paper, we presented an idea called 'inter-goal sharing' which achieves the full sharing of derivation paths for given goals (observations), and makes the EM learning for PRISM programs more compact and efficient. This implies that we can deal with more observations as training data than ever, and the ML estimates of parameters are expected to be more statistically reliable. Obviously, inter-goal sharing is quite general idea, so there seem to be other mathematical models containing structures to which we can apply the idea of inter-goal sharing. In statistical natural language processing, several probabilistic language model and the related algorithms based on unification-based grammars (e.g. [1, 7, 17]) are proposed for mixing our linguistic knowledge and statistical preference naturally. In particular for parsing, some authors use language models based on log-linear models which allow us to freely handle the constraints among phrases [1, 7]. Recently, dynamic-programming-based methods are proposed for parameter estimation in such log-linear models [6, 9]. These methods use graph structures like explanation graphs, and hence they would benefit from the notion of inter-goal sharing. This is because the methods for parameter estimation in such log-linear models (i.e. maximum entropy methods) also require the independence assumption, described in Section 3. To implement the idea of inter-goal sharing on the other formalisms on first-order logic and probability, it is needed to start investigating the relations on semantics between these formalisms, as done in [11].

REFERENCES

[1] S. Abney, 'Stochastic attribute-value grammars', *Computational Linguistics*, **23**, 597–618, (1997).
[2] N. Angelopoulos, 'Extending the CLP engine for reasoning under uncertainty', in *14th International Symposium on Methodologies for Intelligent Systems (ISMIS2003)*, (2003).
[3] E. Charniak, *Statistical Language Learning*, The MIT Press, 1993.
[4] J. Cussens, 'Parameter estimation in stochastic logic programs', *Machine Learning*, **44**, 245–271, (2001).
[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, 'Maximum likelihood from incomplete data via the EM algorithm', *Journal of the Royal Statistical Society*, **B39**, 1–38, (1977).
[6] S. Geman and M. Johnson, 'Dynamic programming for parsing and estimation of stochastic unification-based grammars', in *Proc. of the 40th Anuual Meeting of the Association for Computational Linguistics (ACL02)*, pp. 279–286, (2002).

unfruitful succeeding derivations. However we note here that, since explanation graphs only contain the successful paths, the running time of the gEM algorithm is not influenced by such unfruitful derivations.

[7] M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler, 'Estimators for stochastic "unification-based" grammars', in *Proc. of the 37th Annual Meeting of the Association for Compuational Linguistics (ACL99)*, pp. 535–541, (1999).
[8] K. Kersting and L. De Raedt, 'Bayesian logic programs', Technical Report 151, University of Freiburg, (2001).
[9] Y. Miyao and J. Tsujii, 'Maximum entropy estimation for feature forests', in *Proc. of Human Language Technology Conference (HLT 2002)*, (2002).
[10] S. Muggleton, 'Stochastic logic programs', in *Advances in Inductive Logic Programming*, ed., L. De Raedt, 254–264, IOS Press, (1996).
[11] A. Puech and S. Muggleton, 'A comparison of stochastic logic programs and Bayesian logic programs', in *Proc. of IJCAI-03 workshop on Learning Statistical Models from Relational Data (SRL2003)*, pp. 121–129, (2003).
[12] L. R. Rabiner, 'A tutorial on hidden Markov models and selected applications in speech recognition', in *Proc. of IEEE*, volume 77, pp. 257–286, (1989).
[13] S. Riezler, *Probabilistic constraint logic programming*, Ph.D. dissertation, Universität Tübingen, 1998.
[14] T. Sato, S. Abe, Y. Kameya, and K. Shirai, 'A separate-and-learn approach to EM learning of PCFGs', in *Proc. of the 6th Natural Language Processing Pacific Rim Symposium (NLPRS-2001)*, pp. 255–262, (2001).
[15] T. Sato and Y. Kameya, 'Parameter learning of logic programs for symbolic-statistical modeling', *Journal of Artificial Intelligence Research*, **15**, 391–454, (2001).
[16] T. Sato and N.-F. Zhou, 'A new perspective of PRISM relational modeling', in *Proceedings of IJCAI-03 workshop on Learning Statistical Models from Relational Data (SRL2003)*, pp. 133–139, (2003).
[17] H. Schmid, 'A generative probability model for unification-based grammars', in *Proc. of the 19th International Conference on Computational Linguistics (COLING 2002)*, (2002).
[18] L. Sterling and E. Sharpiro, *The Art of Prolog*, The MIT Press, 1986.
[19] H. Tamaki and T. Sato, 'OLD resolution with tabulation', in *Proc. of the 3rd International Conference on Logic Programming (ICLP86)*, pp. 84–98, (1986).
[20] H. Tanaka, T. Takezawa, and J. Etoh, 'Japanese grammar for speech recognition considering the MSLR method', in *SIG Notes on Spoken Language Processing*, number 015-025. Information Processing Society of Japan, (1997). In Japanese.
[21] N. Uratani, T. Takezawa, H. Matsuo, and C. Morita, 'ATR integrated speech and language database', Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, (1994). In Japanese.
[22] D. S. Warren, 'Memoing for logic programs', *Communications of the ACM*, **35**(3), (1992).
[23] N.-F. Zhou and T. Sato, 'Efficient fixpoint computation in linear tabling', in *Proc. of the 5th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP 03)*, pp. 275–283, (2003).
[24] N.-F. Zhou, T. Sato, and K. Hashida, 'Toward a high-performance system for symbolic and statistical modeling', in *Proc. of IJCAI-03 workshop on Learning Statistical Models from Relational Data (SRL2003)*, pp. 153–159, (2003).