Using WFSTs for Efficient EM Learning of Probabilistic
CFGs and Their Extensions

Yoshitaka Kameya†, Takashi Mori† and Taisuke Sato†

Probabilistic context-free grammars (PCFGs) are a widely known class of probabilistic language models. The Inside-Outside (I-O) algorithm is well known as an efficient EM algorithm tailored for PCFGs. Although the algorithm requires inexpensive linguistic resources, there remains a problem in its efficiency. This paper presents an efficient method for training PCFG parameters in which the parser is separated from the EM algorithm, assuming that the underlying CFG is given. A new EM algorithm exploits the compactness of well-formed substring tables (WFSTs) generated by the parser. Our proposal is general in that the input grammar need not take Chomsky normal form (CNF) while it is equivalent to the I-O algorithm in the CNF case. In addition, we propose a polynomial-time EM algorithm for CFGs with context-sensitive probabilities, and report experimental results with the ATR dialogue corpus and a hand-crafted Japanese grammar.

Key Words: Probabilistic context-free grammars, EM algorithm, Inside-Outside algorithm, Well-formed substring tables

1 Introduction

Probabilistic context-free grammars (PCFGs) are a widely known class of probabilistic language models. They can be seen as context-free grammars (CFGs) in which each production rule is associated with a real number, interpreted as a probability or a parameter. The probability of a sentence or its parse is computed from these rule probabilities and exploited in various predictive tasks. However, practical problems, such as cost and subjectivity, arise if we manually specify the rule probabilities. One solution for this is to train PCFGs, i.e., to automatically estimate the rule probabilities from corpora.

From syntactically annotated corpora, it is straightforward to estimate rule probabilities as the relative frequencies of rules in the corpora. However, we use unbracketed (only morphologically analyzed) corpora as a less expensive training resource. In the literature, the Inside-Outside algorithm (hereafter the I-O algorithm) (Baker 1979; Lari and Young 1990) is a well-known
method for training PCFGs from such unbracketed corpora. We can regard the I-O algorithm as an EM algorithm (Dempster, Laird, and Rubin 1977) tailored for PCFGs, as it is built on a triangular matrix, that was originally used in the Cocke-Younger-Kasami (CYK) parser. The I-O algorithm is certainly a polynomial-time algorithm, but its cubic computation time hinders us from handling large-scale corpora. Furthermore, it has a limitation in applicability, in that the underlying CFG must take Chomsky normal form (CNF).

To overcome these shortcomings, we propose an efficient method for training PCFG parameters by assuming that the underlying CFG is given. In the proposed method, we introduce well-formed substring tables (WFSTs), which are data structures originally used in efficient parsing algorithms. The entire process of the proposed method is split into the following two steps:

**Parsing:** Using an efficient parser, we first obtain all parses of all sentences in the corpus, where the parses are kept implicitly in the form of a WFST.

**EM learning:** We then extract a data structure called support graphs from the WFST, and run the graphical EM algorithm (gEM), an EM algorithm tailored for the support graphs. WFST is a generic name for data structures that contain all partial parse trees obtained during parsing (Tanaka 1988; Nagata 1999). Using WFSTs is a standard technique for preventing the parser from re-analyzing phrases that have already been analyzed. The parsers finally output full parses by assembling the partial parses in a WFST. Table 1 lists WFSTs used in well-known parsers. Fujisaki, Jelinek, Cocke, Black, and Nishino (1989) used parse information in PCFG training, as we do in our proposal; however, they did not exploit WFSTs.

The proposed method makes substantial improvement in generality and efficiency at the same time. To be more concrete, it has a couple of advantages:

**Advantage 1:** The proposed method is a generalization of previous EM learning methods for PCFGs, such as the I-O algorithm and Fujisaki et al.’s method.

**Advantage 2:** Given a practical grammar, the proposed method runs significantly (by orders of magnitude, in our experiments) faster than the I-O algorithm.

**Advantage 3:** The proposed method is a generalization of polynomial-time EM algorithms for

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Examples of WFSTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parser</td>
<td>WFST</td>
</tr>
<tr>
<td>CYK</td>
<td>Triangular matrix</td>
</tr>
<tr>
<td>Earley</td>
<td>Earley charts</td>
</tr>
<tr>
<td>Generalized LR</td>
<td>Packed shared parse forest</td>
</tr>
</tbody>
</table>
“context-free grammars with context-sensitive probability” (Magerman and Weir 1992), such as the rule bigram models (Kita, Morimoto, Ohkura, Sagayama, and Yano 1994). As stated before, the I-O algorithm works on a triangular matrix, which is the WFST of the CYK parser, so it reduces to the proposed algorithm, where the CYK parser and the gEM algorithm are cascaded (Advantage 1). On the other hand, the proposed algorithm does not require the assumption that the underlying CFG takes CNF when combining the Earley parser or the generalized LR (GLR) parser (Tomita and Ng 1991) with the gEM algorithm. The paper will also show that the proposed method includes Stolcke’s probabilistic Earley parser (Stolcke 1995) and a training method from bracketed corpora, proposed by Pereira and Schabes (1992). Furthermore, Advantage 2 comes from the fact that the proposed method only scans a compact data structure, i.e., a WFST. A combination with the GLR parser would further reduce the training time, thanks to the parser’s bottom-up nature and pre-compilation of CFGs into LR tables. Advantage 3 exhibits a benefit from the generality of the proposed method, and in this paper, we present a polynomial-time EM algorithm for Kita et al.’s (1994) rule bigram models.

The rest of the paper is outlined as follows. First, Section 2 formally introduces PCFG, the CYK parser, the I-O algorithm, and their related notions. Then, Section 3 describes a combination of the CYK parser and the gEM algorithm, and compares the result with the I-O algorithm to see Advantage 1. To examine Advantage 2, Section 4 reports an experimental result where the training time of a combination of the GLR parser and the gEM algorithm is measured using the ATR dialogue corpus (SLDB). Section 5 shows Advantage 3 specifically, by presenting a polynomial-time EM algorithm for an extension of PCFG. Lastly, Section 6 describes related work and provides an additional discussion on Advantage 1. Most of the example grammars and sentences, and their parsing results are borrowed from (Nagata 1999), possibly with some modifications.

2 Preliminaries

In this section, we introduce some concepts and notation related to the paper. First, we let $A, B, \ldots$ be non-terminal symbols in a CFG, and $a, b, \ldots$ terminal symbols. Also, $\rho$ indicates a non-terminal or terminal symbol, and $\zeta, \xi, \nu$ indicate an empty sequence or sequences that comprise non-terminal and terminal symbols. An empty sequence is denoted by $\varepsilon$. The symbols in example grammars are written in typewriter fonts (e.g., $S, NP, \ldots$). A list whose $n$-th element

---

1 To be precise, the proposed method includes Pereira and Schabes’s method when an underlying CFG is given.
is denoted by $y_n$ is represented by $\langle y_1, y_2, \ldots \rangle$. For a list $Y = \langle \ldots, y, \ldots \rangle$, we can say $y \in Y$. The cardinality of a set $X$, the number of symbols in a sequence $\zeta$, and the number of elements in a list $Y$ are denoted by $|X|$, $|\zeta|$, and $|Y|$, respectively.

### 2.1 Probabilistic context-free grammars

First, we define a CFG $G$ by a quadruple $\langle V_n, V_t, R, S \rangle$, where $V_n$ is a set of non-terminal symbols, $V_t$ is a set of terminal symbols, $R$ is a set of production rules, and $S$ is a starting symbol ($S \in V_n$). A production rule $r$ in $R$ takes the form $A \rightarrow \zeta$ and replaces a non-terminal symbol $A$ appearing in some sequence $\zeta'$ with $\zeta$. In this paper, we always apply a production rule that replaces the leftmost non-terminal symbol, i.e., we focus only on leftmost derivations. We write $\zeta \overset{r}{\Rightarrow} \xi$ when $\zeta$ is rewritten into $\xi$ by a production rule $r$. When such rewritings are performed for zero or more times, we write $\zeta \overset{*}{\Rightarrow} \xi$ and say that $\xi$ is derived from $\zeta$. If we emphasize that there is one or more rewritings, we write $\zeta \overset{+}{\Rightarrow} \xi$. A sequence $w$ of terminal symbols that can be derived from the start symbol $S$ (i.e., $S \overset{*}{\Rightarrow} w$) is called a sentence. The set of sentences that can be derived using the production rules in a CFG $G$ is called the language of $G$, and is denoted by $L_G$.

Then, we denote by $G(\theta)$ a PCFG whose underlying CFG is $G$. Here, $\theta$ is a $|R|$-dimensional vector, and is called the parameters of the PCFG. Each element in $\theta$ is referred to by $\theta(r)$, where $r \in R$, and we assume that $0 \leq \theta(r) \leq 1$ and $\sum_{\zeta: (A \rightarrow \zeta) \in R} \theta(A \rightarrow \zeta) = 1$. With a PCFG, the rules applied are supposed to be chosen independently. Thus, in a derivation $\zeta_0 \overset{r_1}{\Rightarrow} \zeta_1 \overset{r_2}{\Rightarrow} \zeta_2 \overset{r_3}{\Rightarrow} \ldots \overset{r_K}{\Rightarrow} \zeta_K$, the generative probability $P(r|\theta)$ of a sequence $r = \langle r_1, r_2, \ldots, r_K \rangle$ of the applied rules is computed by

$$P(r|\theta) = \prod_{k=1}^{K} \theta(r_k).$$

(1)

Letting $\sigma(r, r)$ be the number of occurrences of a rule $r$ in rule applications $r$, the probability above can be computationally simplified as

$$P(r|\theta) = \prod_{r \in R} \theta(r)^{\sigma(r, r)}. \tag{2}$$

Furthermore, let $\psi(w)$ be a set of possible rule applications to generate a sentence $w$. Then, noting that $w$ is uniquely determined given rule applications $r$, we have the following relation concerning a joint probability $P(w, r|\theta)$:

$$P(w, r|\theta) = \begin{cases} P(r|\theta) & \text{if } r \in \psi(w), \\ 0 & \text{otherwise}. \end{cases} \tag{3}$$
From this, we have the generative probability $P(w|\theta)$ of a sentence $w$ being derived from the start symbol $S$ as follows:

$$P(w) = \sum_{r \in R} P(w, r|\theta) = \sum_{r \in \Psi(w)} P(r|\theta). \quad (4)$$

If parameter $\theta$ is obvious from the context, we abbreviate $P(r, \ldots|\theta)$ and $P(w, \ldots|\theta)$ as $P(r, \ldots)$ and $P(w, \ldots)$, respectively. In addition to the independence of rule applications, any PCFG $G(\theta)$ in this paper is assumed to satisfy the following conditions:

- $G(\theta)$ is consistent, i.e., $\sum_{w \in L_G} P(w|\theta) = 1$ holds.
- $G$ has no $\varepsilon$ rule, i.e., no production rule whose right hand side is $\varepsilon$.
- There is no cyclic production w.r.t. $G$, i.e., $G$ has no nonterminal $A$ such that $A \Rightarrow A$.

Chi and Geman (1998) proved that, given an underlying CFG satisfying the last two conditions and unbracketed corpora $C$ of finite-length sentences, PCFG $G(\theta^*)$ is consistent where $\theta^*$ is the parameters trained by the I-O algorithm.

### 2.2 Corpora and parse trees

In a sentence $w = w_1 w_2 \cdots w_n \in L_G$, each $w_j$ is called a word ($n > 0$, $j = 1 \ldots n$). Then, we introduce Earley-style word positions $d = 0 \ldots n$ in $w$. For $0 \leq d \leq d' \leq n$, a subsequence or a phrase $w_{d+1} \cdots w_{d'}$ between positions $d$ and $d'$ is denoted by $w_{d,d'}$ (note that $w = w_{0,n}$). A phrase $w_d \cdots w_{d'}$ may be written by a list $\langle w_d, \ldots, w_{d'} \rangle$. For a sentence $w \in L_G$, a parse tree of $w$ represents a derivation process $S \Rightarrow w$ in a tree form. Since we focus on leftmost derivations, a parse tree $t$ of $w$ is uniquely determined from rule applications $r$ in $S \Rightarrow w$, and thus, we will refer to $t$ and $r$ interchangeably. Having assumed that there is neither $\varepsilon$ rule nor cyclic production, a subtree $t'$ of a parse tree $t$ of a sentence $w$ is uniquely identified by a pair $\langle d, d' \rangle$ where the words in $w_{d,d'}$ are leaf nodes in $t'$. From this observation, letting $A$ be the root node of $t'$, we often refer to $t'$ by a label $A(d, d')$, called a subtree label. Then, a parse tree $t$ of a sentence $w$ can be seen as a set $\mathcal{L}(t)$ of such subtree labels of non-leaf nodes. $\mathcal{L}(t)$ is called a label set of $t$. Some may find that a pair $\langle d, d' \rangle$ of word positions corresponds to a bracket in bracketed corpora. We define $\mathcal{B}(t) \overset{\text{def}}{=} \{ \langle d, d' \rangle \mid A(d, d') \in \mathcal{L}(t) \}$ and call $\mathcal{B}(t)$ the bracket set of $t$.

Furthermore, suppose that a production rule $A \rightarrow \rho_1 \rho_2 \cdots \rho_M$ is applied in a tree $t$, and $\rho_1, \rho_2, \ldots, \rho_M$ are the root nodes of subtrees $\rho_m(d_{m-1}, d_m)$ ($m = 1 \ldots M$), as illustrated in Figure 1. Then, we introduce a partial order $@$ such that $A(d_0, d_M)@\rho_m(d_{m-1}, d_m)$, and say that $A(d_0, d_M)$ is a parent of $\rho_m(d_{m-1}, d_m)$, or inversely, $\rho_m(d_{m-1}, d_m)$ is a child of $A(d_0, d_M)$.

521
With a slight adjustment in notation, we jointly write the partial orderings above as

\[ A(d_0, d_M) \rho_1(d_0, d_1) \rho_2(d_1, d_2) \cdots \rho_M(d_{M-1}, d_M) \]  

and call this a parent-children pair of subtrees. We define \( T(t) \) as the entire collection of such parent-children pairs in a tree \( t \). For a sequence \( r \) of rule applications that corresponds to a parse tree \( t \), we define \( L(r) \), \( B(r) \), and \( T(r) \) as:

\[ L(r) = L(t), \quad B(r) = B(t), \quad T(r) = T(t). \]

We consider four types of corpora for training PCFGs: (1) labeled corpora, (2) fully bracketed corpora, (3) partially bracketed corpora, and (4) unbracketed corpora. Our training scheme is maximum likelihood estimation, where a corpus \( C = \langle c_1, c_2, \ldots, c_N \rangle \) of \( N \) sentences is considered as the result of \( N \) independent samplings (i.e., probabilistic derivations) using the rules with probabilities in PCFG \( G(\theta) \). Letting \( w_\ell \) and \( r_\ell \) be, respectively, the sentence and the rule applications obtained in the \( \ell \)-th sampling (\( \ell = 1 \ldots N \)), \( c_\ell = \langle w_\ell, L(r_\ell) \rangle \) when \( C \) is a labeled corpus, \( c_\ell = \langle w_\ell, B(r_\ell) \rangle \) when \( C \) is a fully bracketed corpus, \( c_\ell = \langle w_\ell, B_\ell \rangle \) when \( C \) is a partially bracketed corpus, and \( c_\ell = w_\ell \) when \( C \) is an unbracketed corpus (\( w_\ell \in L_G, r_\ell \in \psi(w_\ell), \quad B_\ell \subseteq B(r_\ell) \)).

### 2.3 The CYK parser

The CYK parser is applicable to the CFGs in CNF. We prepare an \( (n_\ell \times n_\ell) \) upper triangular matrix \( T^{(\ell)} \) for a sentence \( w_\ell \) in an unbracketed corpus \( C \) (\( n_\ell = |w_\ell| \)). Unlike a typical formulation of the CYK parser, the row numbers of the triangular matrix are decremented by one in order to maintain consistency with the Earley-style word positions we use. The element \( T_{d,d'} \) at the \( d \)-th row and the \( d' \)-th column in the triangular matrix stores all partial parse trees for the phrase \( w_{d,d'} \).

Figure 2 shows the routine CYK-PARSER, which implements the CYK parser. Starting from the diagonal elements in the triangular matrix, we build up partial parse trees (parent-children
2.4 The Inside-Outside algorithm

As mentioned before, we aim to train the parameters of a PCFG from a given corpus \( \mathcal{C} = (c_1, c_2, \ldots, c_N) \), following a standard manner of maximum likelihood estimation. For a labeled corpus \( \mathcal{C} \), the relative frequency \( \theta^*(r) \) of a rule \( r \) is exactly the maximum likelihood estimate of the parameter \( \theta(r) \). Generally, however, such labeled corpora are expensive, and it seems
Fig. 4 Triangular matrix for $G$ and a sentence (急いで, 走る, 一郎, を, 見た)

Fig. 5 Two full parse trees (numbers at the bottom are word positions)

more likely that only unbracketed corpora are available. The I-O algorithm, an EM algorithm tailored for PCFGs, is used in such a case, since the relative frequency method cannot be applied to unbracketed corpora. The I-O algorithm is a maximum likelihood estimation method, i.e., it finds the parameters $\theta^*$ that locally maximize the likelihood $\prod_{\ell=1}^N P(w_\ell \mid \theta)$ or its logarithm $\sum_{\ell=1}^N \log P(w_\ell \mid \theta)$ (called the log-likelihood), given an unbracketed corpus $C = (w_1, w_2, \ldots, w_N)$.

In the literature including Lari and Young (1990)'s work, the rule set $R$ of an underlying CFG is not given, while only set $V_t$ of terminals and set $V_n$ of non-terminals are given. For comparison, we describe the I-O algorithm to which some rule set $R$ is given. Indeed, the I-O algorithm given $V_t$ and $V_n$ is equivalent to the one given the following rule set (abbreviated as $R_{\text{max}}$ hereafter):

$$R_{\text{max}}(V_n, V_t) \overset{\text{def}}{=} \{ A \to BC \mid A, B, C \in V_n \} \cup \{ A \to a \mid A \in V_n, a \in V_t \}.$$  \hspace{1cm} (6)

Note that, in both cases, the rule set needs to take CNF. While Lari and Young (1990) used the I-O algorithm for learning set $R$ of production rules as well, we focus on the training of parameters $\theta$.

The central part of the I-O algorithm is the computation of two types of probabilities: the
inside probabilities $P(A \Rightarrow w_{d,d'}^{(\ell)})$ and the outside probabilities $P(S \Rightarrow w_{0,d}^{(\ell)}A w_{d',n_\ell}^{(\ell)})$ ($\ell = 1 \ldots N$, $A \in V_n$, $0 \leq d < d' \leq n_\ell$). These probabilities are stored into the array variables $\beta_{d,d'}^{(\ell)}[A]$ and $\alpha_{d,d'}^{(\ell)}[A]$, respectively. These array variables are prepared in the element $T_{d,d'}$ in the triangular matrix. The generative probability $P(S \Rightarrow w_\ell)$ of a sentence $w_\ell$ is stored into $\beta_{0,n_\ell}^{(\ell)}[S]$.

The routines Get-Beta and Get-Alpha, which are used for computing the inside and outside probabilities, respectively, are presented in Figure 6. In these routines, for simplicity, we assume that the array variables $\alpha_{d,d'}^{(\ell)}[\cdot]$ and $\beta_{d,d'}^{(\ell)}[\cdot]$ are initialized as zero, whenever the routines are called. Starting from the diagonal elements of the triangular matrix, Get-Beta computes the inside probabilities towards the top-right corner $\beta_{0,n_\ell}^{(\ell)}[\cdot]$, just like the CYK parser builds up partial parse trees. Conversely, Get-Alpha starts from the top-right corner $\alpha_{0,n_\ell}^{(\ell)}[\cdot]$ and computes the outside probabilities towards the diagonal elements. This way of computing the inside and outside probabilities can be seen as dynamic programming. After having computed the inside and outside probabilities, the conditional expected occurrences (the expected rule counts hereafter) of $A \rightarrow BC$ and $A \rightarrow a$ in rule applications, given a corpus $C$, are computed as:

```
1: procedure Get-Beta() begin
2:   for $\ell := 1$ to $N$ do begin
3:     for $d := 0$ to $n_\ell - 1$ do /* Compute the inside probabilities for diagonal elements */
4:       foreach $A$ such that $(A \rightarrow w_{d+1}^{(\ell)}) \in R$ do
5:         $\beta_{d+1}^{(\ell)}[A] := \theta(A \rightarrow w_{d+1}^{(\ell)});$
6:       for $k := 2$ to $n_\ell$ do /* Compute the inside probabilities for non-diagonal elements */
7:         for $d := 0$ to $n_\ell - k$ do
8:           foreach $A \in V_n$ do
9:             $\beta_{d+k}^{(\ell)}[A] := \sum_{B,C:(A \rightarrow BC) \in R} \theta(A \rightarrow BC) \sum_{k' = 1}^{k-1} \beta_{d+k+k'}^{(\ell)}[B] \beta_{d+k'}^{(\ell)}[C]$;
10:         end
11:     end;  
12:   end;
13: procedure Get-Alpha() begin
14:   for $\ell := 1$ to $N$ do begin
15:     $\alpha_{0,n_\ell}^{(\ell)}[S] := 1; /* Specially initialize the outside prob. for S at the top-right corner */$
16:     for $k := n_\ell$ downto $2$ do
17:       for $d := 0$ to $n_\ell - k$ do
18:         foreach $B \in V_n$ do
19:           $\alpha_{d+k}^{(\ell)}[B] := \sum_{A,X:(A \rightarrow BX) \in R} \theta(A \rightarrow BX) \sum_{k' = k+1}^{n_\ell-d} \alpha_{d+k+k'}^{(\ell)}[A] \beta_{d+k+k'}^{(\ell)}[X]$
20:           + $\sum_{A',Y:(A' \rightarrow YB) \in R} \theta(A' \rightarrow YB) \sum_{k' = 1}^{d} \alpha_{d-k'\,d+k+k'}^{(\ell)}[A'] \beta_{d-k'}^{(\ell)}[Y]$;
21:         end
22:     end
23:   end
```

Fig. 6 Routines for computing the inside probabilities (above) and outside probabilities (below)
\[ \eta[A \rightarrow BC] := \sum_{\ell=1}^{N} \frac{1}{\beta_{0,n_{\ell}}[S]} \sum_{k=2}^{n_{\ell}} \sum_{d=0}^{k-1} \sum_{k'=1}^{k} \theta(A \rightarrow BC) \alpha_{d,d+k}[A] \beta_{d,d+k'}[B] \beta_{d+k',d+k}[C], \]  
(7)

\[ \eta[A \rightarrow a] := \sum_{\ell=1}^{N} \frac{1}{\beta_{0,n_{\ell}}[S]} \sum_{d=0}^{n_{\ell}-1} \theta(A \rightarrow a) \alpha_{d,d+1}[A]. \]  
(8)

Furthermore, parameters \( \theta(A \rightarrow \zeta) \) are re-estimated from the expected rule counts above:

\[ \theta(A \rightarrow \zeta) := \eta[A \rightarrow \zeta] / \sum_{\zeta' : (A \rightarrow \zeta') \in R} \eta[A \rightarrow \zeta']. \]  
(9)

In the I-O algorithm, we first initialize \( \theta \) randomly, and then iteratively update \( \theta \) by Get-Beta; Get-Alpha; and Eqs. 7, 8, and 9. With this iteration, the log-likelihood \( \sum_{\ell=1}^{N} \log P(w_{\ell}) = \sum_{\ell=1}^{N} \log \beta_{0,n_{\ell}}[S] \) increases monotonically and finally converges. After convergence, the I-O algorithm terminates and outputs the parameters at the same time as the trained ones.

Here, we evaluate the complexity of the I-O algorithm. In general, the number of iterations is unknown in advance since it depends on the initial parameters, so we evaluate the complexity of the I-O algorithm with that of one iteration. Given a set \( V_n \) of nonterminals and a set \( V_t \) of terminals, the worst-case complexity is measured considering the case with \( R = R_{\text{max}}(V_n, V_t) \).

Let \( L \) be the length of the longest sentence in a training corpus \( C \). Then, by examining the for or foreach loops and the ranges of the summations in Get-Beta and Get-Alpha (Figure 6), the worst-case complexity of the I-O algorithm is \( O(|R|L^3N) = O(|V_n|^3L^3N) \).

### 2.5 Additional notes on the Inside-Outside algorithm

The most expensive part in the I-O algorithm is the computation of the inside and outside probabilities. That is, at Line 9 in Get-Beta, we compute the inside probabilities, taking into account all the situations illustrated in Figure 7 (1). For the outside probabilities, the first and second terms on the right hand side at Lines 7 and 8 in Get-Alpha correspond to all the situations in Figure 7 (2) and (3), respectively. One may see that the process of probability computation in the I-O algorithm is similar to top-down parsers, since it considers all possible
situations without taking into account the input sentences \( w_\ell \). In general, ignoring the constraints from the input sentences slows down the parser, and therefore, by analogy, the I-O algorithm’s top-down nature seems to produce an extra computational cost.

To see this more formally, we revisit the derivation of the I-O algorithm described in Lafferty (1993).\(^2\) First, let us define \( \sigma(r,r) \) as the occurrences of a rule \( r \) in rule applications \( r \), as we did in Section 2.1. Then, the I-O algorithm can be obtained by first considering the computation of the expected rule count \( \eta[r] \) of a rule \( r \):

\[
\eta[r] = \sum_{\ell=1}^{N} \sum_{r} P(r|w_\ell) \sigma(r,r) = \sum_{\ell=1}^{N} \frac{1}{P(w_\ell)} \sum_{r} P(w_\ell, r) \sigma(r,r),
\]

second by rewriting Eq. 10 into the form using the partial derivative w.r.t. parameter \( \theta(r) \):

\[
\eta[r] = \theta(r) \cdot \sum_{\ell=1}^{N} \frac{1}{P(w_\ell)} \frac{\partial P(w_\ell)}{\partial \theta(r)} = \theta(r) \cdot \sum_{\ell=1}^{N} \frac{1}{P(w_\ell)} \frac{\partial}{\partial \theta(r)} \sum_{r} P(w_\ell, r),
\]

and finally by rewriting Eq. 11 into a dynamic programming style. Specifically, substituting \( r = (A \rightarrow BC) \) into Eq. 11, the I-O algorithm computes \( \frac{\partial}{\partial \theta(A \rightarrow BC)} \sum_{r} P(w_\ell, r) \) as follows (here subscripts \( \cdot_\ell \) and superscripts \( \cdot_{\ell'} \) are omitted):

\[
\frac{\partial}{\partial \theta(A \rightarrow BC)} \sum_{r} P(w, r) = \frac{\partial}{\partial \theta(A \rightarrow BC)} \sum_{r \text{ s.t. } A \rightarrow BC \text{ appears in } r} P(w, r) = \frac{\partial}{\partial \theta(A \rightarrow BC)} \sum_{d,k,k'} \sum_{r \text{ s.t. } A \rightarrow BC \text{ appears in } r \text{ with the position } (d,d+k',d+k)} P(w, r),
\]

\[
= \frac{\partial}{\partial \theta(A \rightarrow BC)} \sum_{d,k,k'} P(S \Rightarrow w_{0,d}Aw_{d+k,n}) \theta(A \rightarrow BC) \cdot P(B \Rightarrow w_{d,d+k'}) \sum_{d,k,k'} P(C \Rightarrow w_{d+k',d+k}) \Rightarrow w_{d+k',d+k}).
\]

The transformation in Eq. 12 is done independently of the input sentence \( w \) or its parse tree \( t \in \psi(w) \); therefore, the I-O algorithm runs in a top-down manner.

Another approach to computing the expected rule counts \( \eta[r] \) is just to transform Eq. 10 into Eq. 14 below using Eq. 3, where the parse information \( \psi \) is exploited directly. In this approach,

\(^2\) Lafferty (1993) describes the case where the corpus size \( N = 1 \).
ψ is obtained in advance using some efficient parser. Now, we see that Eq. 14 is the same as Fujisaki et al.’s (1989) method in our notation.

\[
\eta[r] = \sum_{\ell=1}^{N} \frac{1}{P(w_\ell)} \sum_{r \in \psi(w_\ell)} P(r) \sigma(r, r) \tag{14}
\]

By using Eq. 14, unlike the I-O algorithm, we can avoid probability computations that are not related to ψ. However in general, |ψ(w)| is exponential to the sentence length |w|, and thus, it is not feasible to compute Eq. 14 as it is. The proposed method, which is presented next, introduces the notion of dynamic programming like the I-O algorithm, and computes Eq. 14 quickly, using a WFST held in some efficient parser. Thus, we can say that the proposed algorithm harmonizes the advantages of Fujisaki et al.’s method and the I-O algorithm.

3 Proposed method

The outline of the proposed method is illustrated in Figure 8. As inputs, the proposed method is given an underlying CFG \( G = \langle V_n, V_t, R, S \rangle \) of the target probabilistic CFG \( G(\theta) \) and an unbracketed corpus \( C \). Then, it returns trained parameters \( \theta \) as the output. In the proposed method, we split the entire training process into two steps: parsing and EM learning. First, we analyze each sentence \( w_\ell \) in the corpus \( C \) by some efficient parser. Then, the parse information is stored into a WFST of the parser. This parse information is collectively equivalent to a set \( \psi(w_\ell) \) of parse trees of \( w_\ell \), but is stored fragmentarily. Therefore, we need an extraction step.
for the fragmentary parse information after parsing, and the extracted data structure is called a support graph. Finally, based on the support graphs, we run the gEM algorithm and return \( \theta \) as trained parameters.

In the case of a CFG \( G1 \) and a sentence \( w_\ell = \langle \text{急いで, 走る, 一郎, を, 見た} \rangle \) in Figure 3, a support graph is extracted from the parent-children pairs marked with \( \circ \) and \( \bullet \) in Figure 4. As illustrated in this example, the support graphs scanned by the gEM algorithm can be much smaller than the entire triangular matrix, and accordingly, we can gain a significant speed-up over the I-O algorithm which inherently scans the entire matrix.

### 3.1 Preliminaries

As a preparation, we introduce some notations. In what follows, we will work for \( \ell = 1 \ldots N \). First, we define \( T_\ell \stackrel{\text{def}}{=} \bigcup_{r \in \psi(w_\ell)} T(r) \) and \( V_\ell \stackrel{\text{def}}{=} \bigcup_{r \in \psi(w_\ell)} L(r) = \{ \tau \mid \tau @ \cdot \in T_\ell \} \).\(^3\) We then define \( O_\ell \stackrel{\text{def}}{=} \langle \tau_1, \tau_2, \ldots, \tau_{|V_\ell|} \rangle \) where the elements in \( O_\ell \) are the members in \( V_\ell \) and totally ordered so that \( \tau_k @ \tau_{k_1} \tau_{k_2} \cdots \tau_{k_M} \in T_\ell \Rightarrow k < k_1, k_2, \ldots, k_M \) holds. Furthermore, we introduce:

\[
\tilde{\psi}_\ell(A(d, d')) \stackrel{\text{def}}{=} \left\{ \begin{array}{l}
E \mid A(d, d') @ \rho_1(d_0, d_1) \rho_2(d_1, d_2) \cdots \rho_M(d_{M-1}, d_M) \in T_\ell, \\
E = \{ \rho_m(d_{m-1}, d_m) \mid m = 1 \ldots M \} \\
\cup \{ A \rightarrow \rho_1 \rho_2 \cdots \rho_M \}, \quad d = d_0, \ d' = d_M
\end{array} \right\}.
\]

\( T_\ell \) is a set of parent-children pairs of subtrees in the parse trees of \( w_\ell \). We also define \( V_\ell \) as a set of subtree labels in the parse trees of \( w_\ell \). \( O_\ell \) is an ordered set of members in \( V_\ell \), totally ordered satisfying the partial order @ in \( T_\ell \). The first element \( \tau_1 \) of \( O_\ell \) is always \( S(0, n_\ell) \). \( \tilde{\psi}_\ell \) represents a logical relationship among subtrees and production rules. For example, we interpret

\[
\tilde{\psi}_\ell(A(d, d')) = \{ \{ A \rightarrow B_1 C_1, \ B_1(d, d'_1), \ C_1(d'_1, d') \}, \{ A \rightarrow B_2 C_2, \ B_2(d, d'_2), \ C_2(d'_2, d') \} \},
\]
as the following statement:

To build up a subtree \( A(d, d') \) for a sentence \( w_\ell \), there are only two ways. In the first way, we apply a rule \( A \rightarrow B_1 C_1 \) and build subtrees \( B_1(d, d'_1) \) and \( C_1(d'_1, d') \). In the second way, we apply a rule \( A \rightarrow B_2 C_2 \) and build subtrees \( B_2(d, d'_2) \) and \( C_2(d'_2, d') \).

\( O_\ell \) and \( \tilde{\psi}_\ell \) jointly constitute a support graph described in the next subsection.

Figure 5 shows two parse trees \( t1 \) and \( t2 \) of a sentence \( w_\ell = \langle \text{急いで, 走る, 一郎, を, 見た} \rangle \) given a CFG \( G1 \) in Figure 3. Let \( r_1 \) and \( r_2 \) be the corresponding rule applications, respectively. Then, we have \( \psi(w_\ell) = \{ r_1, r_2 \} \), and \( T_\ell \) is obtained as follows:

\(^3\) We have defined \( L(r) \) and \( T(r) \) in Section 2.2.
\[ T_\ell = T(r_1) \cup T(r_2) \]
\[ = \{ S(0,5)@PP(0,4)V(4,5), PP(0,4)@NP(0,3)P(3,4), NP(0,3)@VP(0,2)N(2,3), \]
\[ VP(0,2)@ADV(0,1)\psi_{(1,2)}, V(4,5)@見た (4,5), P(3,4)@を (3,4), N(2,3)@一郎 (2,3), \]
\[ V(1,2)@走る (1,2), ADV(0,1)@急いで (0,1) \} \]
\[ \cup \{ S(0,5)@ADV(0,1)VP(1,5), ADV(0,1)@急いで (0,1), VP(1,5)@PP(1,4)V(4,5), \]
\[ PP(1,4)@NP(1,3)P(3,4), NP(1,3)@V(1,2)N(2,3), V(1,2)@走る (1,2), \]
\[ N(2,3)@一郎 (2,3), P(3,4)@を (3,4), V(4,5)@見た (4,5) \} \]

\[ O_\ell \] is not determined uniquely in general, but it is certain that \( S(0,5) \) is the first element. For example, we have:
\[ O_\ell = \{ S(0,5), VP(1,5), PP(1,4), NP(1,3), V(4,5), PP(0,4), P(3,4), \]
\[ NP(0,3), N(2,3), VP(0,2), V(1,2), ADV(0,1) \} \]

We also show \( \tilde{\psi}_\ell \) according to the order of \( O_\ell \):
\[ \tilde{\psi}_\ell(S(0,5)) = \{ S@PP V, PP(0,4), V(4,5) \}, \quad \tilde{\psi}_\ell(NP(0,3)) = \{ NP@VP N, \]
\[ VP(0,2), N(2,3) \} \]
\[ \tilde{\psi}_\ell(VP(1,5)) = \{ VP@PP V, PP(1,4), V(4,5) \}, \quad \tilde{\psi}_\ell(N(2,3)) = \{ N@一郎 \} \]
\[ \tilde{\psi}_\ell(PP(1,4)) = \{ PP@NP P, NP(1,3), P(3,4) \}, \quad \tilde{\psi}_\ell(V(1,2)) = \{ V@走る \} \]
\[ \tilde{\psi}_\ell(NP(1,3)) = \{ NP@V N, NP(1,2), N(2,3) \}, \quad \tilde{\psi}_\ell(V(4,5)) = \{ V@見た \} \]
\[ \tilde{\psi}_\ell(V(0,4)) = \{ PP@NP P, NP(0,3), P(3,4) \}, \quad \tilde{\psi}_\ell(NP(0,3)) = \{ NP@VP N, \]
\[ VP(0,2), N(2,3) \} \]
\[ \tilde{\psi}_\ell(P(3,4)) = \{ P@を \} \]
\[ \tilde{\psi}_\ell(ADV(0,1)) = \{ \text{ADV@急いで} \} \]

### 3.2 Support graphs

It would be easier to understand the gEM algorithm if we represent the pair \( \langle O_\ell, \tilde{\psi}_\ell \rangle \) as a data structure called a support graph \( \Delta_\ell \). Indeed, the word “graphical” in the name comes from this viewpoint. First, Figure 9 (a) shows the support graph that corresponds to the example of \( O_\ell \) and \( \tilde{\psi}_\ell \) in the last subsection. A support graph \( \Delta_\ell \) is a directed acyclic graph that has a structure similar to a recursive transition network (RTN), and consists of disconnected subgraphs \( \tilde{\Delta}_\ell(\tau) \) where \( \tau \in O_\ell \). Each \( \tilde{\Delta}_\ell(\tau) \) is called a support subgraph for \( \tau \) and labeled by \( \tau = A(d,d') \). Each \( \tilde{\Delta}_\ell(\tau) \) has two special nodes—called the starting node and the ending node—labeled by start and end in Figure 9, respectively. For each \( E \in \tilde{\psi}_\ell(\tau) \), the starting node, the nodes labeled by an element in \( E \) (a rule \( A \to \zeta \) or a subtree \( A(d,d') \)), and the ending node are connected in this order. Note that two or more nodes can have the same label. The path from the starting node to the ending node is called the local path, which is also referred to by \( E \). In a local path, the nodes labeled by a rule \( A \to \zeta \) are called basic nodes and those labeled by a subtree \( A(d,d') \) are called intermediate nodes. These nodes are denoted by \( \bigcirc \) and \( \bigcirc \), respectively, as in
sequence of rule applications
the rule labels of the basic nodes visited, where the nodes are ordered as in Figure 9, we have a
the ending node of the support subgraph of
we can find all possible sequences of rule applications in
a traversal, we have a label set that means one parse tree of the sentence
the possible destinations. If we collect the subtree labels of the intermediate nodes visited during
visiting an intermediate node
is used in justifying the gEM algorithm (Appendix A). Here, we show an example of a recursive
Figure 9. Support graphs have the following features:
(1) We can conduct a recursive traversal over a support graph \( \Delta_\ell \), like the way over an RTN.
(2) There are multiple ways of traversal that share some partial paths.
(3) For each \( E \in \tilde{\psi}(\tau) \) in a support subgraph \( \tilde{\Delta}_\ell(\tau) = \langle \tau, \tilde{\psi}(\tau) \rangle \), \( \tau@\tau' \) holds for every \( \tau' = A(d,d') \in E \).
(4) The numbers of basic and intermediate nodes in a local path are not predefined.
A recursive traversal, the first feature, is performed as follows. We first start from the starting
node of a support subgraph \( S(0, n_\ell) \) and visit the nodes one by one along the direct edges. When
visiting an intermediate node \( \tau = A(d,d') \), we jump to the starting node of the support subgraph
labeled by \( \tau \). Next, when reaching the ending node of the current support subgraph, we go back
to the original node. After repeating such recursive visits, the traversal finishes when we reach
the ending node of the support subgraph of \( S(0, n_\ell) \). In a branch at some node, we choose one of
the possible destinations. If we collect the subtree labels of the intermediate nodes visited during
a traversal, we have a label set that means one parse tree of the sentence \( w_\ell \). Similarly, collecting
the rule labels of the basic nodes visited, where the nodes are ordered as in Figure 9, we have a
sequence of rule applications \( r \in \psi(w_\ell) \) in the leftmost derivation of \( w_\ell \). By exhaustive traversals,
we can find all possible sequences of rule applications in \( \psi(w_\ell) \). The notion of recursive traversal
is used in justifying the gEM algorithm (Appendix A). Here, we show an example of a recursive
traversal in Figure 9 (b), where the path of the traversal is drawn as a dotted line.
The second feature above is obtained because, in a recursive traversal, we may jump into the same support subgraph \( \tilde{\Delta}_\ell(\tau) \) from two or more intermediate nodes labeled with \( \tau \). This structure sharing compresses support graphs, and accordingly, the gEM algorithm efficiently computes various probability values. For example, we jump into subgraph \( \tilde{\Delta}_\ell(V(4,5)) \) from the nodes labeled with \( V(4,5) \) and marked with \( \times \) in Figure 9 (a).

The third feature is obvious from the assumption on underlying CFGs that we have neither \( \varepsilon \) rule nor cyclic production \( A \Rightarrow A \), and from the definitions of \( O_\ell \) and \( \tilde{\psi}_\ell \). In other words, if \( \tau \subord \tau' \), then the nodes in the support subgraph \( \tilde{\Delta}_\ell(\tau') \) of \( \tau' \) never refer to \( \tau \). Based on this, the gEM algorithm works in a generalized dynamic programming fashion in computing the inside and outside probabilities (Section 2.4). Finally, the fourth feature shows the generality of support graphs, which is fully exploited by the gEM algorithm.

### 3.3 Extracting support graphs

We next explain how to extract a support graph from the WFST held in the parser. As mentioned before, \( O_\ell \) is an ordered set of members in \( V_\ell \) such that the total order in \( O_\ell \) satisfies the partial order \( \subord \) in the WFST \( T_\ell \). In general, a total order that satisfies a given partial order can be found by topological sorting, and thus we conduct topological sorting to obtain \( O_\ell \) and pick up \( \tilde{\psi}_\ell \) during sorting. Figure 10 (above) shows a routine Extract-CYK for extracting support graphs in this way. Extract-CYK is rather general, while its subroutine should be tailored for the WFSTs of the parser in use. Figure 10 (below) shows a subroutine Visit-CYK that runs over a triangular matrix of the CYK parser.

For these routines, we first prepare a stack\(^4\) \( U \) and a flag array \( Visited[\cdot] \) in a global area. Then, we call Visit-CYK recursively to visit the subtrees \( A(d,d') \) from the top-right corner of the triangular matrix (Line 5 in Extract-CYK). After returning from all recursive calls, we push the current subtree label into the stack \( U \) (Line 10 in Visit-CYK). During recursive calls, we record \( \tilde{\psi}_\ell \) as well (Lines 3 and 6 in Visit-CYK). Note that we put traces into the flags \( Visited[\cdot] \) to avoid revisits (Lines 2, 7, and 8 in Visit-CYK). Finally, \( O_\ell \) is obtained by popping up the subtree labels from \( U \) (Lines 6 and 7 in Extract-CYK).

The GLR parser does not require CNF for the underlying CFG; hence, for packed shared forests, we need to introduce a more general routine than Visit-CYK. However, the basic structure should be the same in that we use a stack, a flag array for traces and recursive calls. Finally, the routines for extracting support graphs often resemble a routine for outputting or

\(^4\) For stack operation, we prepare ClearStack\((U)\), which clears the stack \( U \); PushStack\((x,U)\), which pushes an object \( x \) into \( U \); and PopStack\((U)\), which returns the popped object from \( U \).
counting full parse trees, often provided in the parser software, and so such a built-in routine can be a base for implementation.

### 3.4 Graphical EM algorithm

The proposed method’s main routine \textsc{Learn-PCFG} is presented in Figure 11. We have described two subroutines \textsc{CYK-Parser} and \textsc{Extract-CYK}, and in this subsection, we present \textsc{Graphical-EM}, which implements the gEM algorithm. Similar to the I-O algorithm, the central part of the gEM algorithm is computing the inside and outside probabilities. The inside and outside probabilities of each \( \tau \in O_\ell \) are, respectively, stored into the array variables \( P[\ell, \tau] \) and \( Q[\ell, \tau] \), kept in support subgraph \( \tilde{\Delta}_\ell(\tau) = \langle \tau, \tilde{\psi}_\ell(\tau) \rangle \). \( \tilde{\Delta}_\ell(\tau) \) has an array variable \( R[\ell, \tau, E] \) for

```
1: procedure Extract-CYK() begin
2:   for \( \ell := 1 \) to \( N \) do begin
3:     Initialize all \( \tilde{\psi}_\ell(\cdot) \) to \( 0 \) and all \textit{Visited}[] to \texttt{NIL};
4:     CLEARStack(U); /* Initialize the stack */
5:     VISIT-CYK(\( \ell, S, 0, n_\ell \)); /* Parser-specific recursive routine; Traverse from top-right */
6:     for \( k := 1 \) to \( |U| \) do \( \tau_k := \textit{PopStack}(U) \); /* Extract the sorting result from the stack */
7:     \( O_\ell := \{ \tau_1, \tau_2, \ldots, \tau_{|U|} \} \)
8:   end
9: end.
10: procedure VISIT-CYK(\( \ell, A, d, d' \)) begin
11:   Let \( \tau = A(d, d') \) and then \textit{Visited}[\( \tau \)] := \texttt{YES}; /* Put the trace */
12:   if \( d' = d + 1 \) \textbf{and} \( A(d, d')@w_\ell(d, d') \in T_{d,d'}^{(\ell)} \) then Add a set \( \{ A \rightarrow w_\ell \} \) to \( \tilde{\psi}_\ell(\tau) \)
13:   else
14:     foreach \( A(d, d')@B(d, d'')C(d'', d') \in T_{d,d'}^{(\ell)} \) do begin
15:       Add a set \( \{ A \rightarrow BC, B(d, d''), C(d'', d') \} \) to \( \tilde{\psi}_\ell(\tau) \);
16:     if \textit{Visited}[\( B(d, d'') \)] = \texttt{NIL} then VISIT-CYK(\( \ell, B, d, d'' \)); /* Recursion */
17:     if \textit{Visited}[\( C(d'', d') \)] = \texttt{NIL} then VISIT-CYK(\( \ell, C, d'', d' \)) /* Recursion */
18:   end;
19:   PUSHStack(\( \tau, U \))
20: end.
21: end.
```

**Fig. 10** Routines for extracting support graphs: Extract-CYK (above) and Visit-CYK (below)

```
1: procedure Learn-PCFG(\( \mathcal{C} \)) begin
2:   CYK-Parser(\( \mathcal{C} \)); /* Generate WFSTs as a parsing result for \( \mathcal{C} \) */
3:   Extract-CYK(); /* Extract support graphs from the WFSTs */
4:   Graphical-EM(); /* Train the parameters \( \theta \) based on the support graphs */
5: end.
```

**Fig. 11** Main routine \textsc{Learn-PCFG} for training PCFGs
each local path \( E \in \tilde{\psi}_\ell(\tau) \). We also have an array variable \( \eta[A \to \zeta] \) that stores the expected rule count of each rule \( A \to \zeta \). **Graphical-EM** has two subroutines: **Get-Inside-Probs**, which computes the inside probabilities, and **Get-Expectations**, which simultaneously computes the outside probabilities and the expected rule counts.

**Graphical-EM** is shown in Figure 12. In **Graphical-EM**, we first initialize all parameters (Line 2). Then, we iterate **Get-Inside-Probs**, **Get-Expectations** and re-estimation of parameters in this order (Lines 7 and 8). After the convergence of log-likelihood \( \lambda \) (Line 12), we consider the parameters \( \theta \) at the moment as the trained ones \( \theta^* \). The log-likelihood is computed using the generative probability \( P(w_\ell) \) of \( w_\ell \) stored in \( P[\ell, S(0, n_\ell)] \) (Lines 4 and 11). Figure 13 shows two subroutines **Get-Inside-Probs** and **Get-Expectations**. Figure 14 illustrates how these subroutines work over an example support graph.

The inside probabilities \( P[\ell, \tau] \) in **Get-Inside-Probs** are computed from the last support subgraph according to \( O_\ell \). For each local path \( E \in \tilde{\psi}_\ell(\tau_k) \) in \( \tau_k \)'s support subgraph \( \Delta_\ell(\tau_k) = \langle \tau_k, \tilde{\psi}_\ell(\tau_k) \rangle \) \((k = 1 \ldots |O_\ell|)\), we compute the product of inside probabilities of the nodes in the local path and store the product into \( R[\ell, \tau_k, E] \) (Lines 7 and 8 and Figure 14 (1)). In computing the product, we multiply the parameter \( \theta(A \to \zeta) \) for a basic node \( A \to \zeta \) or the inside probability \( P[\ell, \tau'] \) for an intermediate node \( \tau' \) (Figure 14 (2)). Finally, we compute \( P[\tau_k] \) by summing

```
1: procedure Graphical-EM() begin
2: initialize all parameters \( \theta(A \rightarrow \zeta) \) such that \( P(w_\ell|\theta) > 0 \) for all \( \ell = 1 \ldots N \);
3: Get-Inside-Probs();
4: \( \lambda^{(0)} := \sum_{\ell=1}^{N} \log P[\ell, S(0, n_\ell)] \);
5: repeat
6: Get-Expectations();
7: foreach \( (A \rightarrow \zeta) \in R \) do
8: \( \theta(A \rightarrow \zeta) := \eta[A \rightarrow \zeta] / \sum_{\zeta' : (A \rightarrow \zeta') \in R} \eta[A \rightarrow \zeta'] \);
9: \( m := 1 \);
10: Get-Inside-Probs();
11: \( \lambda^{(m)} := \sum_{\ell=1}^{N} \log P[\ell, S(0, n_\ell)] \)
12: until \( \lambda^{(m)} - \lambda^{(m-1)} \) becomes sufficiently small
end.
```

---

5 From the third feature of support graphs (Section 3.2), letting \( \tau' = \tau_{k'} \), we always have \( k < k' \). Also note that the computation of \( P \) is conducted from the last support subgraph according to \( O_\ell \). Then, it is obvious that \( P[\ell, \tau'] \) has already been computed when it is referred to.
Fig. 13 Two subroutines Get-Inside-Probs (above) and Get-Expectations (below)

Fig. 14 Probability computation in Get-Inside-Probs (left) and Get-Expectations (right)
\( \mathcal{R}(\ell, \tau_k, E) \) (Line 10 and Figure 14 (3)).

On the other hand, inversely with Get-Inside-Probs, Get-Expectations starts from the first support subgraph according to \( O_\ell \). In this routine, we first initialize the array variables \( Q \) and \( \eta \). In particular, regarding the outside probabilities \( Q[\ell, \cdot] \), we set one for \( \tau_1 = S(0, n_\ell) \), which is the first element of \( O_\ell \), and set zero for the others (Lines 5 and 6). Next, for each \( k = 1 \ldots |O_\ell| \), we consider a local path \( E \) in the support subgraph \( \tilde{\Delta}_k(\tau_k) \) of \( \tau_k \) (Line 8), and the subtree \( \tau' \in E \) whose outside probability \( Q \) is to be updated at Line 11. Then, the product of the local outside probability of \( \tau' \) in \( E \) (the product of inside probabilities of the nodes other than \( \tau' \)) and the outside probability of \( \tau_k \) (the parent of \( \tau' \)) is accumulated into \( Q[\ell, \tau'] \) (Figure 14 (4)). However, at Line 10, for a basic node \( A \rightarrow \zeta \), the product of the probability \( \mathcal{R}(\ell, \tau_k, E) \) of the local path \( E \) and the outside probability \( Q[\ell, \tau_k] \) of the parent \( \tau_k \), divided by the generative probability \( P(w_\ell) \) of the sentence \( w_\ell \),\(^7\) is accumulated into \( \eta[A \rightarrow \zeta] \) (Figure 14 (5)). After such asynchronous accumulations into \( \eta[A \rightarrow \zeta] \), \( \eta[A \rightarrow \zeta] \) contains the expected rule count of rule \( A \rightarrow \zeta \) when Get-Expectations terminates. The computations in the \( gEM \) algorithm described above are justified with the notion of recursive traversal over support graphs (Section 3.2). Such a justification will be made in Appendix A.

In general, the EM algorithm is a hill-climbing algorithm targeting the log-likelihood function, and hence, only guarantees a local maximum likelihood estimate. As a result, the quality of the trained parameters depends heavily on the initial parameters. Lari and Young (1990) proposed to initialize the parameters using trained hidden Markov models. One simple solution for local optimality is to first repeat the EM algorithm \( h \) times with random initial parameters, and then to pick up the parameters in convergence at the highest log-likelihood as the best trained parameters. In this paper, this method is called random restarting.

\(^6\) Letting \( \tau' = \tau_{k'} \), \( k < k' \) always holds from the third feature of support graphs, and so \( \tau' \) appears after \( \tau_k \) in the order \( O_\ell \). Inversely, in a support subgraph \( \tilde{\Delta}_k(\tau_{k''}) \) where \( k'' = k \ldots |O_\ell| \), \( Q[\ell, \tau_k] \) will never be modified, and hence the computation of \( Q[\ell, \tau_k] \) in the right hand side of the substitution at Line 11 has been completed.

\(^7\) As shown at Line 2 in GRAPHICAL-EM, we initialize \( \theta \) so that \( P(w_\ell|\theta) > 0 \) holds for all \( \ell = 1 \ldots N \). So, after any re-estimations of \( \theta \) in the \( gEM \) algorithm, it never happens that \( P(w_\ell|\theta) = 0 \). Since, as roughly proved in Appendix A, the expected rule counts \( \eta[r] \) in the \( gEM \) algorithm and those in Fujisaki et al.’s (1989) method (Eq. 14) are always equal, this property can be proved inductively as follows. First, suppose that \( P(w_\ell|\theta^{(m)}) > 0 \) holds under the parameters after the \( m \)-th re-estimation. Then, there should exist some \( r \in \phi(w_\ell) \) that satisfies \( P(r|\theta^{(m)}) > 0 \). Furthermore, for any rule \( r \in r, \eta[r] > 0 \) should hold under \( \theta^{(m)} \), from the fact that \( \sigma(r, r) > 0 \) and Eq. 14. This implies that the parameter \( \theta^{(m+1)}(r) \) after the next re-estimation is also positive, as seen from Line 8 in GRAPHICAL-EM. Again, for \( r \) considered above, we have \( P(r|\theta^{(m+1)}) > 0 \), and consequently, \( P(w_\ell|\theta^{(m+1)}) > 0 \). Now, we summarize that \( P(w_\ell|\theta^{(m)}) > 0 \Rightarrow P(w_\ell|\theta^{(m+1)}) > 0 \), and therefore, if we initialize the parameters as \( \theta^{(0)} \) such that \( P(w_\ell|\theta^{(0)}) > 0 \), then \( P(w_\ell|\theta^{(m)}) > 0 \) after the subsequent re-estimations (\( m = 1, 2, \ldots \)). [Q.E.D.] To make sure that \( P(w_\ell|\theta) > 0 \) in practical cases, it suffices to choose \( \theta \) s.t. \( \theta(r) > 0 \) holds for every \( r \in R \).
3.5 Probabilistic parsing

Having trained the parameters, we can find the most likely derivation \( \mathbf{r}_\ell^* \) by
\[
\mathbf{r}_\ell^* := \text{argmax}_{\mathbf{r}} P(\mathbf{r}| \mathbf{w}_\ell) = \text{argmax}_{\mathbf{r} \in \psi(\mathbf{w}_\ell)} P(\mathbf{r})
\]
for each input sentence \( \mathbf{w}_\ell \) in the test corpus. We define \( t^*_\ell \) as the parse tree corresponding to \( \mathbf{r}_\ell^* \), and resolve the syntactic ambiguity in \( \mathbf{w}_\ell \) by \( t^*_\ell \).

Since \( |\psi(\mathbf{w}_\ell)| \) is still exponential here, so we aim to find \( t^*_\ell \) on the basis of support graphs.

Figure 15 shows a routine \textsc{Predict} for finding \( t^*_\ell \), and its subroutines \textsc{Get-Max-Probs} and \textsc{Construct-Tree}.\(^8\) \textsc{Predict} takes as input the corpus \( \mathcal{C} = \langle \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_N \rangle \), and for each

1: \begin{verbatim}
  procedure \textsc{Predict}(\mathcal{C}) begin
  let \( N = |\mathcal{C}| \);
  \textsc{CYK-Parse}(\mathcal{C}); \textsc{Extract-CYK}(); \textsc{Get-Max-Probs}();
  for \( \ell := 1 \) to \( N \) do begin
    \( \mathcal{L}_\ell^* := \emptyset \);
    \textsc{Construct-Tree}(\ell, \tau_1) /* We always have \( \tau_1 = S(0, n_\ell) */
  end
end.
\end{verbatim}

1: \begin{verbatim}
  procedure \textsc{Get-Max-Probs}() begin
  for \( \ell := 1 \) to \( N \) do begin
    let \( \mathcal{O}_\ell = \langle \tau_1, \tau_2, \ldots, \tau_{|\mathcal{O}_\ell|} \rangle \);
    for \( k := |\mathcal{O}_\ell| \) down to 1 do begin
      foreach \( E \in \tilde{\psi}_\ell(\tau_k) \) do begin
        \( \mathcal{R}[\ell, \tau_k, E] := 1 \);
        foreach \( \tau' \in E \) do
          if \( \tau' = (A \rightarrow \zeta) \) then \( \mathcal{R}[\ell, \tau_k, E] *\theta(A \rightarrow \zeta) \) else \( \mathcal{R}[\ell, \tau_k, E] *= \mathcal{P}[\ell, \tau'] \)
        end /* foreach \( E \) */
      end /* foreach \( E \) */
    end /* for \( k \) */
  end /* for \( \ell \) */
end.
\end{verbatim}

1: \begin{verbatim}
  procedure \textsc{Construct-Tree}(\ell, \tau) begin
  foreach \( \tau' \in \delta[\ell, \tau] \) such that \( \tau' = A(d, d') \) do begin
    add \( \tau' \) to \( \mathcal{L}_\ell^* \);
    \textsc{Construct-Tree}(\ell, \tau')
  end
end.
\end{verbatim}

Fig. 15 Routine \textsc{Predict} for finding the most likely parse and its two subroutines

---

\(^8\) In the original Japanese version of this paper, \textsc{Predict} does not work correctly, since it uses the inside probabilities computed by \textsc{Get-Inside-Probs}.\n
sentence \(w_ℓ\), it outputs a set \(L(ℓ^*_*)\) of subtree labels in the most likely parse tree. In PREDICT, we first run a parser, a routine for extracting support graphs, and a subroutine GET-MAX-PROBS (Line 3). GET-MAX-PROBS determines the most likely subtrees fragmentarily in a dynamic programming fashion. Another subroutine, CONSTRUCT-TREE, is then called for building the most likely, full parse tree from such fragmentary parse information (Line 6).

Here, we describe further details. GET-MAX-PROBS works similarly to GET-INSIDE-PROBS, but has two differences. First, to find the most likely local paths, GET-MAX-PROBS uses the maximization operator instead of the summation at Line 10, i.e., the meanings of two array variables, \(P\) and \(R\), have been changed. The second difference is that GET-MAX-PROBS uses an additional array variable \(δ[ℓ, τ_k]\) for recording the most likely local path itself (Line 11). CONSTRUCT-TREE conducts a recursive traversal over \(δ[ℓ, τ_k]\) and adds the subtree labels \(A(d, d')\) in the most likely local path \(δ[ℓ, τ]\) into \(L_ℓ^*\) (Line 3). This process corresponds to building the most likely parse tree. If we extend \(δ[ℓ, τ]\) to contain two or more candidate local paths, we would have top-\(n\) most likely parse trees.

### 3.6 Complexity

As stated before, since the number of iterations depends on the initial parameters, we evaluate the complexity of the I-O algorithm with that of one iteration. This is exactly the complexity inside the repeat loop in GRAPHICAL-EM. First, let \(O_ℓ = \langle τ_1^{(ℓ)}, τ_2^{(ℓ)}, ..., τ_{|O_ℓ|}^{(ℓ)} \rangle\) for each \(ℓ = 1...N\). Note that, in the routine GET-INSIDE-PROBS called from GRAPHICAL-EM, we visit each element in \(\tilde{ψ}_ℓ(τ_κ^{(ℓ)})\) once for all \(k = 1...|O_ℓ|\). Then, the complexity of GET-INSIDE-PROBS is evaluated as \(O(μ_{num}μ_{maxsize}N)\), where:

\[
μ_{num} = \max_{ℓ=1...N} \sum_{k=1}^{|O_ℓ|} |\tilde{ψ}_ℓ(τ_κ^{(ℓ)})|, 
\]

\[
μ_{maxsize} = \max_{E: ℓ=1...N, k=1...|O_ℓ|, E∈\tilde{ψ}_ℓ(τ_κ^{(ℓ)})} |E|. 
\]

Similarly we see that GET-EXPECTATIONS requires \(O(μ_{num}μ_{maxsize}N)\) time.

Let \(L\) be the maximum sentence length in a training corpus \(C\), and consider a set \(V_n\) of nonterminals and a set \(V_t\) of terminals. Here, we evaluate the worst-case complexity with the maximum grammar \(R_{max}\) (Eq. 6) in CNF. For such a grammar, we have

\[
\tilde{ψ}_ℓ(A(d, d')) = \{ A→BC, B(d, d''), C(d'', d') \mid B, C \in V_n, d < d'' < d' \}
\]

for all \(A, d\) and \(d'\), such that \(A \in V_n, 0 ≤ d, d'' ≤ L\) and \(d + 2 ≤ d'\) (the case with \(d' =
\(d + 1\) is ignorable). Then, since \(|O_\ell| = |\{A(d, d') \mid A \in V_n, 0 \leq d < d' \leq L\}| = O(|V_n|^2L)\) and \(|\tilde{\psi}_\ell(\tau)| = O(|V_n|^2L)\) hold, \(\mu_{\text{num}} = O(|V_n|^3L^3)\) from the definition in Eq. 16. Also from the definition in Eq. 17, \(\mu_{\text{maxsize}} = 3 = O(1)\). The complexity of parameter re-estimation is \(O(|R_{\text{max}}|) = O(|V_n|^3L^3)\), which is ignorable. From the discussion above, the complexity of the gEM algorithm is \(O(|V_n|^3L^3N)\), i.e., the same as that of the I-O algorithm.

With a grammar in CNF, the worst-case complexity of the CYK parser (CYK-PARSER) and the routine for extracting support graphs (EXTRACT-CYK) is \(O(|V_n|^3L^3N)\), which is the same as that of one EM iteration. However, note that the EM algorithm usually iterates for a few dozens or hundreds of times, so the computation time of these routines is often ignorable in the entire training process. Similarly, assuming CNF, the complexity of computing the generative probability or probabilistic parsing for a sentence is \(O(|V_n|^3L^3)\) \((N = 1)\). For the parsers (e.g., CYK and GLR) that use WFSTs with parent-children pairs of the form in Eq. 5, we have the same support graphs \(\langle O_\ell, \tilde{\psi}_\ell \rangle\) and the same complexity. We evaluate the case with the Earley parser in Appendix B.

4 Experiments on training time

To show Advantage 2 of our proposal, that it can run significantly faster than the I-O algorithm with a practical CFG, we measured the training time for the ATR dialogue corpus (SLDB). The underlying CFG is a modified version of a Japanese grammar \(G^*\) by Tanaka, Takezawa, and Eto (1997), which was originally hand-crafted for speech recognition. The modified grammar has 860 rules, and according to this modified version, the corpus was also modified. \(G^*\) is a CFG whose terminal symbols are fine-grained parts of speech, and has 173 nonterminals and 441 terminals. The average, the minimum, and the maximum sentence length in the corpus are 9.97, 2, and 49, respectively. Since the rule set \(R^*\) in \(G^*\) is not in CNF, we used a GLR parser in the proposed method. However, since the I-O algorithm only works with CFGs in CNF, we transformed \(G^*\) into \(G^*_{\text{Chom}}\) in CNF, which has 2,308 rules, 210 nonterminals, and 441 terminals.

In our experiments, we compare the training time between the proposed method and the I-O algorithm, given \(G^*\) as an underlying CFG. The I-O algorithm is the one described in Section 2.4,

\[9\] More specifically, we cascaded the routine for extracting support graphs and the routine for the gEM algorithm into the MSLR (Morphological and Syntactic LR), which was available from the Tanaka-Tokunaga Laboratory at Tokyo Institute of Technology (http://tanaka-www.cs.titech.ac.jp/pub/mslr/). Although the MSLR parser can perform morphological analysis and syntactic analysis simultaneously, we used only the syntactic analysis part. Including the MSLR parser, the programs used in the experiments are written in C language. We used gcc 2.8.1 as a C compiler. The CPU and OS of the computer we used were Sun UltraSPARC-II 296 MHz and Solaris 2.6, respectively.
and uses rule set $R_{\text{Chom}}^*$ in $G_{\text{Chom}}^*$ instead of $R_{\text{max}}$ (Eq. 6). We measured the time consumed for a re-estimation of the parameters (called the re-estimation time), varying the sentence length $L$. For this, we first gathered the sentences of lengths $L - 1$ and $L$ into a group ($L = 2, 4, \ldots, 26$), and from each group, we randomly picked up 100 sentences as $C_L$. Then, for each $C_L$ being treated as a training corpus, we measured the re-estimation time.

Figure 16 (left) shows the measurement results. The y-axis indicates the average re-estimation time in seconds and $L$ in the x-axis corresponds to the training corpus $C_L$. The curve “Inside-Outside” indicates the re-estimation time in the I-O algorithm, and “IO with pruning” indicates the re-estimation time in a pruning-embedded (or, in short, pruning) version of the I-O algorithm, which is presented by Kita (1999). This modified version is more efficient than the original, in that it skips redundant zero-probability computations in computing outside probabilities. Finally, “Graphical EM” indicates the re-estimation time in the gEM algorithm. To make the graph shapes readable, we magnify and minify the scale of the y-axis of Figure 16 (left) as shown in Figure 16 (center) and Figure 16 (right), respectively.

As shown in Figure 16 (left), the gEM algorithm runs significantly faster than the I-O algorithm and its pruning version. In addition, Figure 16 (center) indicates that the re-estimation time in the I-O algorithm draws a cubic curve w.r.t. the sentence length $L$, as the theory indicates.

Of course, the I-O algorithm using $R_{\text{Chom}}^*$ runs faster than the one using $R_{\text{max}}$, because of the additional constraint from the grammar (e.g., it narrows the range of the outer summation at Line 9 in Get-Beta).

In the experiments, to obtain stable statistics, we did not use 176 sentences that were longer than 26, which account for 1.6% in the corpus.
The pruning version certainly runs faster than the original but only in quadratic time at best, since it still unconditionally scans all elements in the triangular matrix. Noting that we need to repeat re-estimations several hundred times and conduct random restarts, it is not practical to run the I-O algorithm or its pruning version until convergence for the training corpora \( C_L \) where \( L > 20 \).

In contrast, as shown in Fig 16 (right), the proposed method runs almost in linear time in sentence length \( L \), where \( L = 2, 4, \ldots, 26 \). This significant gap from the worst-case complexity \( O(|V_n|^3L^3) \) seems to be brought by the grammatical constraints that greatly reduce the ambiguity of the input sentences, or more specifically, the number of subtrees stored in the WFSTs. At sentence length \( L = 10 \), which is close to the average 9.97 in the ATR dialogue corpus, the proposed method runs about one thousand times faster than the I-O algorithm (about seven hundred times faster than the pruning version).

When we use the random restarting method (Section 3.4) to obtain better parameters w.r.t. the log-likelihood, the entire training time is broken down as follows:

\[
\text{(Entire training time)} = \text{(Parsing time)} + \text{(Time spent on extracting support graphs)} + \text{(Time spent on the gEM algorithm)},
\]

\[
\text{(Time spent on the gEM algorithm)} = \text{(Re-estimation time)} \times \text{(Number of re-estimations until convergence)} \times \text{(Number } h \text{ of random restarts)}.
\]

Using the length-wise training corpora \( C_L \) \( (L = 2, 4, \ldots, 26) \) described before, we measured the breakdowns of training time (the parsing time, the time spent on extracting support graphs and the time spent on the gEM algorithm), and the results are shown in Figure 17. The x-axis and the y-axis respectively indicate the sentence length \( L \) and the consumption time in seconds. Also Figure 17 (left) shows the case without restarts (i.e., \( h = 1 \)) and Figure 17 (right) shows the case with 10 restarts (\( h = 10 \)). We fixed the number of re-estimations until convergence as 100 because it varies depending on the corpus \( C_L \). The parsing time ("Parsing"), the time spent for extracting support graphs ("Support graph") and the time spent on the gEM algorithm ("Graphical EM") are almost linear in sentence length \( L \). Figure 17 (right) also indicates that, when introducing random restarts, the parsing time and the time spent on support-graph extraction are ignorable in the entire training time, as we do not have to repeat parsing and support-graph extraction in random restarts. So these two steps can be seen as a small preprocessing that yields a significant
speed up of training, and in this way, we enjoy the merit of separating the entire training process into parsing and EM learning.

5  EM learning of the extensions of PCFGs

So far, there have been several proposals that incorporate context-sensitivity into PCFGs. However, except for Charniak and Carroll’s (1994) pseudo probabilistic context-sensitive grammars, no EM algorithms have been presented for such extensions of PCFGs. In this section, to show Advantage 3 of the proposed method that it covers polynomial-time EM algorithms for various extensions of PCFGs, we select Kita et al.’s (1994) rule bigram models and present their polynomial-time EM algorithm.

5.1  The rule bigram models

First, we concentrate on leftmost derivations as in the case with PCFGs. Then, relaxing the assumption in PCFGs that production rules are chosen totally independently, we assume that rules are chosen depending on the one at the last choice. By this relaxation, we can add some context-sensitivity, which is not covered by PCFGs, into the rule bigram models. Under this assumption, the generative probability of rule applications $r$ is computed as

$$
P(r) = \theta(r_1 \mid \#) \prod_{k=2}^{K} \theta(r_k \mid r_{k-1}).
$$  (19)
Here, \# is the marker that indicates a sentence boundary, and \(\theta (r \mid r')\) is a parameter associated with each rule \(r \in R\) \((r' \in R \cup \{\#\})\), in which \(\sum_{\zeta; (A \rightarrow \zeta) \in R} \theta (A \rightarrow \zeta \mid r) = 1\) holds for \(A \in V_n\) and \(r \in R \cup \{\#\}\). In Kita et al. (1994), given an unbracketed corpus \(C = \langle w_1, \ldots, w_N \rangle\), a parameter \(\theta (r_k \mid r_{k-1})\) is re-estimated as

\[
\theta^{(m+1)} (r_k \mid r_{k-1}) := \left( \sum_{\ell=1}^{N} \frac{\sum_{r \in \psi (w_\ell)} \sigma (r_{k-1}, r_k; r)}{|\psi (w_\ell)|} \right) / \left( \sum_{\ell=1}^{N} \frac{\sum_{r \in \psi (w_\ell)} \sigma (r_{k-1}; r)}{|\psi (w_\ell)|} \right).
\]

(20)

By the re-estimation in Eq. 21, we can achieve (local) maximum likelihood estimation, but obviously Eq. 21 is not feasible, as \(|\psi (w)|\) is generally exponential in \(|w|\).

5.2 Graphical EM algorithm applied to the rule bigram models

To derive a polynomial-time EM algorithm for the rule bigram models, whose re-estimation is equivalent to Eq. 21, we exploit the generality of the proposed method. Before moving further, we introduce some notations. We first consider the leftmost derivation \(r\) of a sentence \(w\) as follows:

\[
S \Rightarrow \cdots \Rightarrow w_{0,d} A \xi \Rightarrow w_{0,d} A \zeta \xi \Rightarrow \cdots \Rightarrow w_{0,d} A \zeta' \xi \Rightarrow w_{0,d} w_{d,d'} \xi = \langle w_{0,d} \xi \rangle \Rightarrow w
\]

(22)

In Eq. 22, \(r\) is the rule applied just before \(A\) is expanded, and \(r'\) is the last rule applied in a partial derivation \(A \Rightarrow \cdots \Rightarrow w_{d,d'}\). Considering \(r\) and \(r'\) as the context, the subtree that governs \(w_{d,d'}\) is labeled as \(A(d, d'|r, r')\). We refer to rule \(r'\) as \(\text{last}(A, d, d'; r)\), and rule \(r''\), which is applied just after \(r\), by \(A \rightarrow \zeta | r\). In the rule bigram models, we choose \(A \rightarrow \zeta | r\) with probability \(\theta (A \rightarrow \zeta | r)\). We also define \(\text{last}(A, d, d'; w) \overset{\text{def}}{=} \{ \text{last}(A, d, d'; r) \mid r \in \psi (w) \}\), the set of the last applied rules in deriving a subtree \(A(d, d')\) while \(w\) is being generated.

Now, we present an EM algorithm where the CYK parser is adopted in the proposed method. In this case, no change is required for the CYK parser. Furthermore, since the basic control flow of the gEM algorithm can remain the same, the only part we need to work out is the routine
for extracting support graphs. For example, we have the following logical relation \( \tilde{\psi}_\ell \) for \( t_2 \) in Figure 5:

\[
\tilde{\psi}_\ell(\text{VP}(1, 5 \mid \text{ADV} \rightarrow \text{急いで}, \text{V} \rightarrow \text{見た})) = \{ \{ \text{VP} \rightarrow \text{PP} \text{V}_\text{ADV} \rightarrow \text{急いで}, \text{P} \rightarrow \text{を}, \text{V} \rightarrow \text{見た}\} \}.
\]

Here, a subtree label \( A(d, d') \) is literally specialized into \( A(d, d'|r, r') \) by a pair \( \langle r, r' \rangle \) where \( r \) is the rule applied just before the corresponding partial derivation \( A \Rightarrow w_{d,d'} \) occurs, and \( r' \) is the last applied rule in the partial derivation. In the proposed method, context-sensitivity is typically incorporated by literal specialization of subtree labels, as above.

Figure 18 and 19, respectively, show the support-graph extraction routine \( \text{Extract-CYK-RB} \) tailored for the rule bigram models, and its recursive subroutine \( \text{Visit-CYK-RB} \). The subroutine \( \text{Visit-CYK-RB}(\ell, r, A, d, d') \) visits the subtree \( A(d, d') \) in the parse tree of \( w_\ell \), and adds \( \text{last}(A, d, d'; w_\ell) \) into a global array variable \( \text{Last}[A(d, d')] \).

What remains is to modify the gEM algorithm slightly. More specifically, we first replace \( A \rightarrow \zeta, \theta(A \rightarrow \zeta) \) and \( \eta[A \rightarrow \zeta] \) with \( A \rightarrow \zeta|_r, \Theta(A \rightarrow \zeta|_r) \) and \( \eta[A \rightarrow \zeta|_r] \), respectively. Then we wrap the \textbf{foreach} loops at Lines 7 and 8 in \textbf{Graphical-EM} and Line 2 in \textbf{Get-Expectations} by the “\textbf{foreach} \( r \in R \)” loop.

Next, we evaluate the worst-case complexity of the EM algorithm for the rule bigram models. Considering \( R_{\text{max}} \), the worse-case complexity is \( O(|V_n|^2 L^3 N) \). This computational order is quite large, but is still cubic in sentence length \( L \). Furthermore, the significant gap between the worst-case complexity and the actual computation time shown in Section 4 also seems applicable to the rule bigram models. Indeed, Mori, Kameya, and Sato (2000) reported that, with a handcrafted CFG \( G^* \) in Section 4, the proposed method for the rule bigram models only runs about 1.5 times slower than that for PCFGs.

---

12 For simplicity, in Figure 19, we present a somewhat redundant version of \( \text{Visit-CYK-RB} \). That is, it recomputes \( \text{Last}[A(d, d')] \) for the calls with different \( r \)’s.

13 First, for all \( A \in V_n, \langle d, d' \rangle \) such that \( 0 \leq d < d' \leq L \) and \( d + 2 \leq d', \) and \( r, r' \in R \), we have the following support subgraph for \( A(d, d'|r, r') \):

\[
\tilde{\psi}_\ell(A(d, d'|r, r')) = \{ \{ A \rightarrow BC |_r, B(d, d''|A \rightarrow BC, r'') \mid B, C \in V_n, d < d'' < d', r'' \in \text{last}(B, d, d'') \subseteq R \} \},
\]

and \( A(d, d'|r, r') \) appears in \( O_\ell (\ell = 1 \ldots N) \). Then, it is obvious that \( |\tilde{\psi}_\ell(\tau)| = O(|V_n|^2 L|R|) \). In addition, \( O_\ell \) is an ordered set of the members in a subset of \( \{ A(d, d'|r, r') \mid A \in V_n, 0 \leq d < d' \leq L, r, r' \in R \} \), we have \( |O_\ell| = O(|V_n|^2 L^2|R|^2) \). So by definition we have \( \mu_{\text{num}} = O(|V_n|^3 L^3|R|) \) and \( \mu_{\text{maximize}} = O(1) \). Finally, considering the case with \( R = R_{\max} \), the worst-case complexity of the gEM algorithm is \( O(|V_n|^2 L^3|R_{\max}|^3 N) = O(|V_n|^2 L^3 N) \).
6 Related work

In the literature, several probabilistic parsers have been proposed, e.g., Magerman and Marcus’s (1991) Pearly; its successor, Magerman and Weir’s (1992) Picky; and Stolcke’s (1995) probabilistic Earley parser. However, most probabilistic parsers (except Stolcke’s) assume as
input an underlying grammar $G$ together with parameters $\theta$, and do not consider how to train the parameters.

As EM training methods for PCFGs not in CNF, Kupiec’s (1992) method and Stolcke’s probabilistic Earley parser have been proposed. Kupiec’s method first considers PCFGs as recursive transition networks, and trains them on an extended trellis diagram. Kupiec’s method and the I-O algorithm are similar in their top-down approach. WFSTs exploited in the proposed method are essential and well-known in CFG-based parsing methods, so the proposed method seems conceptually simpler than Kupiec’s method, which uses an extended trellis diagram. For PCFGs with neither $\epsilon$ rule nor cyclic production $A \Rightarrow A$, Stolcke’s (1995) method is equivalent to the proposed method, in which the Earley parser and the gEM algorithm are cascaded (Appendix B). Therefore, for such PCFGs, the proposed method is a generalization of Stolcke’s method. In addition, Stolcke did not mention any training method for extensions of PCFGs. It is an interesting future work to extend it to work for PCFGs with $\epsilon$ rules, cyclic productions, or both.

Pereira and Schabes (1992) proposed a method that simultaneously learns the structure and parameters of a grammar, from a partially or fully bracketed corpus. They also showed empirically that the quality of the grammar structure and parameters learned by their method is significantly improved compared to those learned from the corresponding unbracketed corpus. In the proposed method, it is possible to train PCFGs from partially or fully bracketed corpora, just by using a parser that outputs parse trees satisfying the constraints from the brackets. This simplicity comes from a property whereby we only need the WFST for EM learning. The complexity of training PCFGs from a fully bracketed corpus in the proposed method is $O(|V_n|^3L)$, which is the same as that of Pereira and Schabes’s method.

In this paper, we have assumed that an underlying CFG is given. However, automated learning of the grammar structure, or grammar induction, is an important research topic, since it is quite costly for a human to write precise grammars. For instance, given a set $V_n$ of nonterminals and a set $V_t$ of terminals, Lari and Young proposed, first, to run the I-O algorithm with the rule set $R_{\text{max}}(V_n, V_t)$, and second, to remove the rules whose probabilities are sufficiently small (Lari and Young, 1990).

---

14 The MSLR parser, which was used in our experiments (Section 4), has such a functionality.

15 Pereira and Schabes only stated that the complexity is $O(L)$, but it is immediately seen from their description of the algorithm that it is $O(|V_n|^3LN)$ when using a rule set $R_{\text{max}}(V_n, V_t)$. The complexity of the gEM algorithm with fully bracketed corpora is evaluated as follows. First, the size of the set $B(w_\ell)$ of brackets for a sentence $w_\ell$ is $O(|w_\ell|)$, as Pereira and Schabes described. Also note that the parse trees inconsistent with $B(w_\ell)$ cannot be the final parse trees, i.e., cannot be the members of $O_{\ell}$. Then, for each $\ell = 1 \ldots N$, we have $|O_{\ell}| = |\{d,d' \mid A \in V_n \text{ and } (d,d') \in B(w_\ell)\}| = O(|V_n| \cdot |w_\ell|) = O(|V_n|L)$. Since the number of possible $d''$’s in Eq. 18 that are consistent with $B(w_\ell)$ is at most one, $|\tilde{\psi}(A(d,d'))| = O(|V_n|^2)$ holds for each $\ell = 1 \ldots N$. Therefore, we have $\mu_{\text{num}} = O(|V_n|^3L)$ by definition (Eq. 16), and also have $\mu_{\text{max_size}} = O(1)$ as discussed before. Consequently, the complexity of re-estimation in the gEM algorithm is $O(|V_n|^3LN)$. 


Young 1990). Pereira and Schabes’s (1992) method, which has been mentioned before, can also be understood as a method for grammar induction from bracketed corpora. One typical problem in such EM-based approaches is that they only find a local maximum likelihood estimate, and hence the quality of the learned grammar heavily depends on the initial parameters. Against this problem of locality, there have been proposals in training hidden Markov models (HMMs), such as the successive state splitting (SSS) algorithm (Takami and Sagayama 1993) and a structural learning method based on model selection criteria (Ikeda 1995). These methods separate the entire learning process into two steps, training parameters and exploring the model structure, and runs these two steps alternately. When one extends these methods to the structural learning of PCFGs, as a model (grammar) structure is given in the parameter training step, the proposed method will effectively accelerate the learning.

The gEM algorithm was originally proposed by Kameya and Sato (2000) for a probabilistic logic programming language called PRISM (Sato and Kameya 1997). PRISM’s semantic basis is the distribution semantics (Sato 1995), which is a probabilistic extension of the least model semantics in logic programs. The original proposal cascades OLDT (Ordered Linear resolution for Definite clauses with Tabulation) (Tamaki and Sato 1986) and the gEM algorithm. In this paper, we replace OLDT by a CFG parser with a focus on training PCFGs and their extensions. Although OLDT is a generic top-down search technique that is applicable to parsing, the GLR parser runs faster for practical grammars, thanks to pre-compilation of CFGs into LR tables and its bottom-up search strategy. In both cases, since the extracted support graphs are the same, the time spent on the gEM algorithm is also the same.

7 Conclusion

In this paper, we proposed a generic method for training the parameters in PCFGs from unbracketed corpora, under the assumption that the underlying CFG is given. The proposed method improves the I-O algorithm on generality and efficiency (with practical underlying CFGs) at the same time.

The proposed method separates the entire training process into parsing and EM learning. That is, it extracts the parse information from the WFST stored in the parser and exploits the extracted information, which is often compact, in EM training. Using this design, the proposed method overcomes the efficiency of the I-O algorithm, which suffers from its top-down nature. Any technique that improves parsing efficiency would accelerate the proposed method.

We implemented the proposed method and confirmed that, given a hand-crafted Japanese
grammar, it runs significantly (one thousand times at the average sentence length) faster than the I-O algorithm. Based on the generality of the proposed method, we also derived a polynomial-time EM algorithm for CFGs with context-sensitive probabilities (i.e., the rule bigram models) and showed that the proposed method can cover previous methods such as Pereira and Schabes’s (1992) method and Stolcke’s (1995) probabilistic Earley parsers.

In the future, we would like to conduct further experiments with some extensions of PCFGs and work for grammar induction using the proposed method. Also, there is an interesting open problem to derive an EM algorithm for the probabilistic GLR model recently reformulated by Inui, Sornlertlemvanich, Tanaka, and Tokunaga (1998).

Acknowledgement

A revised version of the ATR dialogue corpus and the Japanese grammar used in the experiments were provided by the Tanaka-Tokunaga Laboratory at Tokyo Institute of Technology. We are deeply grateful for their cooperation, and special thanks go to Kiyoaki Shirai at the Tanaka-Tokunaga Laboratory for offering detailed information on the corpus and the grammar, an auxiliary program for preprocessing, and some key papers in the literature. We also thank Nobuhisa Ueda for his insightful comments from the early stage of the work. The work is supported in part by the Discovery Science project, promoted by Grant-in-Aid for Scientific Research on Priority Areas (A).

Reference


Parsin Technologies, pp. 85–94.


Appendix

A Justification of the graphical EM algorithm

In this section, we roughly show that Fujisaki et al.’s (1989) method, the I-O algorithm and the gEM algorithm output the same trained parameters, from the same initial parameters and under the same convergence criterion. For this purpose, we only need to show that these three methods give the same expected rule count \( \eta[A \rightarrow \zeta] \) for any rule \((A \rightarrow \zeta) \in R\). As seen in Section 2.5, the expected rule counts computed by Fujisaki et al.’s method are equal to those computed by the I-O algorithm, so it is sufficient to show that the expected rule counts computed
by the gEM algorithm are equal to those computed by Fujisaki et al.’s method.

First, we consider recursive traversal over a support graph $\Delta_\ell$, which is described in Section 3.2. In what follows, the starting (resp. ending) node in the subgraph of $\tau$ is called “$\tau$’s starting (resp. ending) node.” Now, we consider the traversals in which we collect the rule labels associated with basic nodes, and focus on some basic node $v$ associated with $A \rightarrow \zeta$ in a support graph $\Delta_\ell$. Also, suppose that $v$ is included in a subgraph of $\tau$, and let $E$ be the local path in which $v$ appears. This situation is depicted in Figure 20. Then, we consider all possible recursive traversals in which we start from $S(0, n_\ell)$’s starting node and pass through $v$. Note here that $S(0, n_\ell)$ is the first element of $O_\ell$. We denote by $\psi(v, w_\ell)$ the set of sequences of the rule labels (rule applications) collected in such traversals (obviously $\psi(v, w_\ell) \subseteq \psi(w_\ell)$).

We further introduce some notations. Let $r_1$ be the sequence of the rule labels (partial rule applications) collected in a partial recursive traversal from $\tau$’s starting node to $\tau$’s ending node through the local path $E$ considered above. Also let $r_0$ be a sequence of the rule labels collected in a partial traversal from $S(0, n_\ell)$’s starting node to an intermediate node $u$ which is associated with $\tau$ (Figure 20), and $r_2$ be a sequence of the rule labels collected in a partial traversal from $u$ to $S(0, n_\ell)$’s ending node. Here, we let $\psi_{\text{in}}(v, w_\ell) = \{r_1\}$ and obtain the set $\psi_{\text{out}}(v, w_\ell)$ of possible pairs of $r_0$ and $r_2$ by varying $u$ above. Then, $\psi(v, w_\ell)$ defined above can be seen as a Cartesian product of $\psi_{\text{in}}(v, w_\ell)$ and $\psi_{\text{out}}(v, w_\ell)$.

From the definitions above and the independence assumption in PCFGs, we have:

$$\sum_{r' \in \psi(v, w_\ell)} P(r') = \sum_{(r_0, r_1, r_2) \in \psi(v, w_\ell)} P(r_0, r_1, r_2)$$

$$= \sum_{r_1 \in \psi_{\text{in}}(v, w_\ell)} \sum_{(r_0, r_2) \in \psi_{\text{out}}(v, w_\ell)} P(r_1) P(r_0, r_2)$$

$$= \left( \sum_{r_1 \in \psi_{\text{in}}(v, w_\ell)} P(r_1) \right) \left( \sum_{(r_0, r_2) \in \psi_{\text{out}}(v, w_\ell)} P(r_0, r_2) \right)$$

---

**Fig. 20** Basic node $v$ associated with $A \rightarrow \zeta$, appearing in a support graph $\Delta_\ell$
Examining recursively how $P$ and $R$ are computed in the routine `Get-Inside-Probs`, we see that $R[\ell, \tau, E] = \sum_{r_1 \in \psi_{in}(v, w_\ell)} P(r_1)$. Similarly, examining recursively how $Q$ is computed in `Get-Expectations`, it can be seen that $Q[\ell, \tau] = \sum_{\langle r_0, r_2 \rangle \in \psi_{out}(v, w_\ell)} P(r_0, r_2)$. As a result, we have $Q[\ell, \tau] \cdot R[\ell, \tau, E] = \sum_{r' \in \psi(v, w_\ell)} P(r')$. We then see that the value accumulated into $\eta[A \rightarrow \zeta]$ at Line 10 in `Get-Expectations` is equal to $\frac{1}{P(w_\ell)} \sum_{v: A \rightarrow \zeta \text{ is attached}} \sum_{r' \in \psi(v, w_\ell)} P(r')$.

Disregarding the order of computations, the gEM substantially repeats the computation above for each basic node $v$ associated with $A \rightarrow \zeta$, and for each support graph $\Delta_\ell$ ($\ell = 1 \ldots N$). The expected rule count $\eta[A \rightarrow \zeta]$ is finally computed as:

$$\eta[A \rightarrow \zeta] = \sum_{\ell=1}^{N} \frac{1}{P(w_\ell)} \sum_{v: A \rightarrow \zeta \text{ is attached}} \sum_{r' \in \psi(v, w_\ell)} P(r') . \quad (23)$$

Now, let us consider a recursive traversal from $S(0, n_\ell)$’s starting node, and let $r$ be a sequence of the rule labels (rule applications) collected in the traversal. Then, the number of basic nodes $A \rightarrow \zeta$ we visit in the traversal is exactly $\sigma(A \rightarrow \zeta, r)$ in our notation. So in the summation $\sum_{v: A \rightarrow \zeta \text{ is attached}} \sum_{r' \in \psi(v, w_\ell)}$, a sequence $r \in \psi(w_\ell)$ of rule applications is taken into account for $\sigma(A \rightarrow \zeta, r)$ times. This immediately implies that Eq. 23 is equivalent to Fujisaki et al.’s formula (Eq. 14). From the discussions above, we can conclude that Fujisaki et al.’s method, the I-O algorithm, and the gEM algorithm are equivalent to each other.

**B Relation to Stolcke’s probabilistic Earley parser**

In this section, we roughly relate Stolcke’s probabilistic Earley parser (Stolcke 1995) and the proposed method when the Earley parser and the gEM algorithm are cascaded. First, we briefly describe the probabilistic Earley parser.16

**B.1 Probabilistic Earley parser**

The Earley parser analyzes an input sentence $w_\ell$ based on the set $I_\ell$ of items. Each item is in the form $d'r:A \rightarrow \zeta, \xi$ and indicates (i) $w^{(\ell)}_{0:d'} = w^{(\ell)}_1 \ldots w^{(\ell)}_{d'}$ has been analyzed, (ii) the phrase governed by a nonterminal $A$ starts from the position $d$, (iii) $A$ is expanded by the rule $A \rightarrow \zeta, \xi$, and analysis has been conducted until the point indicated by the dot symbol on the right hand side.

Stolcke’s (1995) probabilistic Earley parser is a natural extension of the Earley parser, where

16 The description is based on Stolcke’s notation with some exceptions to keep consistency with our notation.
each item $d' : dA \rightarrow \zeta, \xi$ is associated with its inside probability $\beta'(dA \rightarrow \zeta, \xi)$.[17] From now on, we refer to such an item as $d' : dA \rightarrow \zeta, \xi [\beta]$ (i.e., $\beta$ a shorthand of $\beta_{d'}(dA \rightarrow \zeta, \xi)$). The inside probability $\beta_{d'}(dA \rightarrow \zeta, \xi)$ is the sum of probabilities of the derivation paths from an item $d : dA \rightarrow \zeta, \xi$ to another item $d' : dA \rightarrow \zeta, \xi$. In the parser, the inside probabilities are computed by three operations:

- **Prediction:** If the current item set $I_\ell$ contains an item $d' : dA \rightarrow \zeta, B \xi [\beta]$ and we have $(B \rightarrow \nu) \in R$, then we add a new item $d' : dB \rightarrow \nu [\beta']$ into $I_\ell$ unless it already exists. Here, we let $\beta' = \theta(B \rightarrow \nu)$.
- **Scanning:** If the current item set $I_\ell$ contains an item $(d' - 1) : dA \rightarrow \zeta, w^{(t)} \xi [\beta]$, then we add a new item $d' : dA \rightarrow \zeta, \xi [\beta']$ into $I_\ell$ unless it already exists. Here, we let $\beta' = \beta$.
- **Completion:** If the current item set $I_\ell$ has two items $d' : dA \rightarrow \zeta, B \xi [\beta]$ and $d'' : dA \rightarrow \zeta, B \xi [\beta]$ such that $d'' < d'$,[18] then we add a new item $d' : dA \rightarrow \zeta, B \xi [\beta']$, where $\beta' = \beta \cdot \beta''$, into $I_\ell$ unless it already exists. If the new item already exists in $I_\ell$, we just increment $\beta'$ by $\beta \cdot \beta''$ (i.e., $\beta' + = \beta \cdot \beta''$).

In EM learning, we require the outside probability $\alpha_{d'}(dA \rightarrow \zeta, \xi)$ for an item $d' : dA \rightarrow \zeta, \xi$. This probability is the sum of probabilities of the paths where we (i) start from the initial item $0_0 : S$, (ii) generate $w^{(t)}_{0,d}$, (iii) pass through $dA \rightarrow \nu \xi$ for some $\nu$, (iv) generate $w^{(t)}_{d',n_\ell}$ starting from $d' \rightarrow \nu \xi$, and (v) finish with $n_\ell : 0 \rightarrow S$. The outside probabilities are obtained by:

- **Reverse Completion:** For each pair $d' : dA \rightarrow \zeta, B \xi [\alpha, \beta]$ in $I_\ell$, we find an item $d' : dA \rightarrow \zeta, B \xi [\alpha', \beta']$ in $I_\ell$ and make two increments: $\alpha' += \beta' \cdot \alpha$ and $\alpha'' += \beta' \cdot \alpha$.

We initialize the outside probability as one for the item $n_\ell : 0 \rightarrow \nu$, and as zero for the other items. After having computed all inside and outside probabilities, we compute the expected rule counts by Eq. 24 and re-estimate the parameters using Eq. 9, as with the I-O algorithm.

$$\eta[A \rightarrow \zeta] = \sum_{\ell=1}^{N} \frac{1}{\beta_{n_\ell}(0 \rightarrow S)} \sum_{(d : dA \rightarrow \zeta) \in I_\ell} \alpha_{d}(dA \rightarrow \zeta) \beta_{d}(dA \rightarrow \zeta)$$ (24)

B.2 Support graphs for the probabilistic Earley parser

To implement the probabilistic Earley parser, we replace EXTRACT-CYK by EXTRACT-EARLEY (Figure 21) in the main routine (Figure 11). We also show the subroutine VISIT-EARLEY.

---

[17] In addition to inside probabilities, (Stolcke 1995) associates a probability called the forward probability with each item. In this paper, however, we omit forward probabilities since they have nothing to do with EM learning.

[18] Stolcke’s method allow the underlying CFG to have $\varepsilon$ rule and cyclic production $A \mp A$. So in his description, the completion operation is also applicable to the case $d'' = d'$. 

553
Fig. 21 Support-graph extraction routine \texttt{Extract-Earley()} tailored for the Earley parser.

\begin{verbatim}
1: procedure \texttt{Extract-Earley()} begin
2: \hspace{1em} for $\ell := 1$ to $N$ do
3: \hspace{2em} Initialize all $\bar{\psi}_\ell(\cdot)$ to $\emptyset$ and all $\text{Visited}[\cdot]$ to $\emptyset$;
4: \hspace{2em} \text{ClearStack}(U);
5: \hspace{2em} \text{Visit-Earley}(\ell, n_{\ell,0} \rightarrow S);
6: \hspace{2em} for $k := 1$ to $|U|$ do $\tau_k := \text{PopStack}(U)$;
7: \hspace{2em} $O_\ell := \{ \tau_1, \tau_2, \ldots, \tau_{|U|} \}$
8: \hspace{1em} end
9: end.
\end{verbatim}

Fig. 22 Subroutine \texttt{Visit-Earley} of \texttt{Extract-Earley}

called from \texttt{Extract-Earley} in Figure 22. In this case, we need not modify the routines for the gEM algorithm. Based on the routine for outputting the full parse trees, \texttt{Extract-Earley} generates support graphs $\Delta_\ell = \langle O_\ell, \bar{\psi}_\ell \rangle$ ($\ell = 1 \ldots N$). Each $\bar{\psi}_\ell$ takes the following form:

\begin{align*}
\bar{\psi}_\ell(d_a \rightarrow B \rightarrow \nu) &= \{ B \rightarrow \nu \}, \quad (25) \\
\bar{\psi}_\ell(d_a \rightarrow \zeta w^{(\ell)}_d \rightarrow \xi) &= \{ (d' - 1):d_a \rightarrow \zeta w^{(\ell)}_d \xi \}, \quad (26) \\
\bar{\psi}_\ell(d_a \rightarrow \zeta B \rightarrow \xi) &= \begin{cases} 
\{ (d'', d_a \rightarrow \zeta B \xi), (d'' : B \rightarrow \nu) \} 
\end{cases} \\
& \quad \text{if } d \leq d'' < d', (d'' : B \rightarrow \nu) \in I_\ell. \quad (27)
\end{align*}
In the gEM algorithm, computing the inside probabilities for a support subgraph in Eq. 25 corresponds to the Prediction operation in the probabilistic Earley parser. Similarly, computing the inside probability for Eq. 26 corresponds to the Scanning operation, and computing the inside (resp. outside) probability for Eq. 27 corresponds to the Completion (resp. Reverse Completion) operation. Last, computing the expected rule counts for Eq. 25 corresponds to Eq. 24.

Now, we evaluate the complexity of the gEM algorithm for the support graphs above. Regarding time consumption, the computation related to the support subgraphs in Eq. 27 occupies the largest part. Given an underlying CFG in CNF, the total number of nodes in such support subgraphs is obtained as \( O(|R|L^3) \) by considering all possible rules and word positions \((d, d', d'')\), where \( R \) is the rule set and \( L \) is the maximum sentence length. When \( R = R_{\text{max}} \), the worse-case complexity of the gEM algorithm is \( O(|V_n|^3L^3) \), which is the same as that of the probabilistic Earley parser.

Here, we consider the case with an underlying CFG not in CNF, and let \( m \) be the maximum number of symbols on the right hand side of a rule. Then, the complexity of the gEM algorithm cascaded after a parser, e.g., the GLR parser, with WFSTs of the form in Eq. 5 is \( O(L^{m+1}) \). On the other hand, when cascaded after the Earley parser, the complexity turns out to be \( O(L^3) \), i.e., it does not depend on \( m \). However, the GLR parser has some advantages such as pre-compilation of CFGs to LR tables and its bottom-up search strategy over the Earley parser, and hence it seems worth choosing an appropriate parser depending on the target grammar.

Yoshitaka Kameya: received his M.S. (1997) and his Ph.D. (2000) in Computer Science from Tokyo Institute of Technology. After working at NS Solutions Corporation, he worked as a research associate (2003–2007) and an assistant professor (2007–2012) at Tokyo Institute of Technology. He has been an associate professor at Meijo University since April 2012. His current interest is in knowledge representation and reasoning in machine learning and data mining. He is a member of the ACM and IEEE.

Takashi Mori: received his M.S. in Computer Science in 2001 from Tokyo Institute of Technology. He joined Sun Microsystems Japan in April 2001.

Taisuke Sato: is a professor of Computer Science department at Tokyo Institute of Technology. He received his M.S. in Electrical Engineering in 1975 and his Ph.D. in Computer Science in 1987 from Tokyo Institute of Technology. His early work includes program transformation and program synthesis based on
fold/unfold transformation in logic programming. He is currently working on
the integration of logical and probabilistic reasoning. Especially he has been
developing a symbolic-statistical modeling language PRISM which is a logic-
based probabilistic programming language for machine learning.

(Received May 10, 2000)
(accepted October 10, 2000)