A Separate-and-Learn Approach to EM Learning of PCFGs
(Revised Feb. 1st, 2001)

Sato, Taisuke† and **Abe, Shigeru**‡ and **Kameya, Yoshitaka**♯ and **Shirai, Kiyoaki**§

†Tokyo Insititute of Technology (`sato@mi.cs.titech.ac.jp`)
‡Tokyo Insititute of Technology (`abe@mi.cs.titech.ac.jp`)
♯ Tokyo Insititute of Technology (`kame@mi.cs.titech.ac.jp`)
§Japan Advanced Institute of Science and Technology (`kshirai@jaist.ac.jp`)

Abstract

We propose a new approach to EM learning of PCFGs. We completely separate the process of EM learning from that of parsing, and for the former, we introduce a new EM algorithm called the *graphical EM algorithm* that runs on a new data structure called *support graphs* extracted from WFSTs (well formed substring tables) of various parsers. Learning experiments with PCFGs using two Japanese corpora indicate that our approach can significantly outperform the existing approaches using the Inside-Outside algorithm (Baker, 1979) and Stolcke's EM algorithm (Stolcke, 1995).

1 Introduction

The objective of this paper is to show that it is possible to realize faster EM learning of PCFGs (probabilistic context free grammars) than the Inside-Outside algorithm (Baker, 1979) and Stolcke's EM algorithm (Stolcke, 1995) by adopting a new EM algorithm[1] called the *graphical EM* algorithm that runs on a new data structure called *support graphs* which can compactly represent parse trees. We assume, unlike (Pereira and Schabes, 1992), that there is a stochastic grammar in addition to an unannotated corpus. This is because the former aims at grammar induction from scratch, whereas we intend to learn parameters for the grammar from the corpus like (Beil et al., 1999; Riezler et al., 2000). Learning experiments using two Japanese corpora show that unsupervised parameter learning of PCFGs (and their extensions) can be significantly sped up. Due to severe space limitations, only PCFGs are treated in this paper. For more details of learning experiments with PCFGs and their extensions such as Pseudo PCSGs (Charniak and Carroll, 1994) and lexicalized PCFGs, the reader is referred to (Sato et al., 2001).[2]

2 Background

Baker proposed the first EM algorithm for PCFGs in the late 70's (Baker, 1979) which is now called the Inside-Outside algorithm (hereafter referred to as the I-O algorithm). It is applicable to PCFG in Chomsky normal form and takes $O(N^3L^3)$ time to update all parameters in one iteration, where N is the number of symbols in a grammar and L the length of an input sentence (Lari and Young, 1990).

Ten years later, Fujisaki et al. proposed another EM algorithm using derivations of a sentence (Fujisaki et al., 1989). Unlike the I-O algorithm, their algorithm incorporates parsing, and hence the blind combination of rules is avoided.

Six years later, Stolcke proposed to use an Earley chart generated by an Earley parser to compute inside and outside probabilities (Stolcke, 1995). The chart is comprised of items augmented with probabilities and inside and outside probabilities are computed from them. His EM algorithm incorporates both parsing and factorized computations of inside and outside probabilities. Thus the redundancies that plagued the I-O algorithm and Fujisaki et al.'s algorithm are eliminated. However there still remain two redundancies. One is that an Earley chart contains items not part of a parse tree but inside probabilities are computed for all of them. The other is that complete items $[d\!:{}_{d'}A \rightarrow \nu.]$ are dynamically combined in every iteration despite the fact that their combinations do not change.

A tacit commonality of these approaches is that they *did not* separate learning from parsing so

[1]The EM algorithm is an algorithm for ML (maximum likelihood) estimation with incomplete data. It iteratively updates parameters so that the likelihood of the observed data increases until it saturates.

[2]Regrettably, it contains certain confusions about the unit of y-axis used in the graphs, but they do not affect the conclusions.

that data structure used for parsing is reused for EM learning, which we believe causes various degrees of inefficiencies. We therefore *completely separate EM learning from parsing.* That is, we first parse input sentences and extract *support graphs* (explained in Section 3) from the WFSTs (well-formed substring tables) such as a triangular table employed by a CYK parser and an Earley chart employed by an Earley parser. They compactly represent all possible derivations of a given sentence as a graph. We then run a new EM algorithm called the *graphical EM algorithm* on the support graphs that efficiently computes inside and outside probabilities in a way of dynamic programming. While the form of support graphs varies with a parser, the graphical EM algorithm does not change and our separate-and-learn approach is experimentally confirmed to bring about a significant speed up of EM learning not only of PCFGs but also of a family of PCFGs such as Pseudo PCSGs and lexicalized PCFGs (Sato et al., 2001).

3 Support graphs

In this section, we introduce *support graphs*, a new data structure for EM learning of stochastic grammars.[3] Let $\mathcal{G}$ be a PCFG and $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(T)}$ a random sample of T sentences. In what follows, R stands for the rule set of $\mathcal{G}$ and $\theta(A \rightarrow \zeta)$ for the parameter associated with a rule $A \rightarrow \zeta \in R$. n_ℓ denotes $|\mathbf{w}^{(\ell)}|$, i.e. the length of $\mathbf{w}^{(\ell)}$ and $\mathbf{w}^{(\ell)}[i,j]$ a substring of $\mathbf{w}^{(\ell)}$ starting at the $i+1$ *th* word and ending at the j *th* word.

In our approach to EM learning of PCFGs, we first parse each sentence $\mathbf{w}^{(\ell)}$ $(1 \leq \ell \leq T)$, obtain a WFST and extract from it a *support graph* Δ_ℓ for $\mathbf{w}^{(\ell)}$. We then run the *graphical EM algorithm* on $\{\Delta_\ell \mid 1 \leq \ell \leq T\}$ to statistically learn parameters associated with $\mathcal{G}$.

Generally speaking, a *support graph* is a graphical representation of a finite set of decision sequences. In the case of PCFGs, a decision is nothing but a probabilistic choice of a rule to expand a non-terminal category. A sequence of decisions uniquely determines a (leftmost) derivation of a sentence, and hence obviously has one-to-one correspondence to a parse tree.

Figure 1 illustrates a support graph for a sentence "astronomers saw stars with ears"[4] extracted from the triangular table generated by a CYK parser. The sentence has two parse trees; [[astronomers]$_{\rm NP}$ [[saw]$_{\rm V}$ [[stars]$_{\rm NP}$ [[with]$_{\rm P}$ [ears]$_{\rm NP}$]$_{\rm PP}$]$_{\rm NP}$]$_{\rm VP}$]$_{\rm S}$ and [[astronomers]$_{\rm NP}$ [[[saw]$_{\rm V}$ [stars]$_{\rm NP}$]$_{\rm VP}$ [[with]$_{\rm P}$ [ears]$_{\rm NP}$]$_{\rm PP}$]$_{\rm VP}$]$_{\rm S}$. As can bee seen, the support graph is a collection of subgraphs labeled like A(i,j) which means that there are partial derivations from non-terminal A that spans from the $i+1$ *th* word to the j *th* word in the input sentence. These subgraphs are ordered linearly, preserving the ancestor-descendent relationship in the parse tree. Subgraphs themselves are comprised of linear graphs. For instance, the subgraph labeled VP(1,5) has two linear graphs corresponding to two successful expansions of VP by (VP→NP VP) and (VP→VP PP). Other nodes have a unique expansion. It is easy to see that the support graph in Figure 1 as a whole represents the two parse trees we mentioned above. We call support graphs of this type *CYK type*.

Formally, a *support graph* Δ_ℓ for $\mathbf{w}^{(\ell)}$ is a *totally*

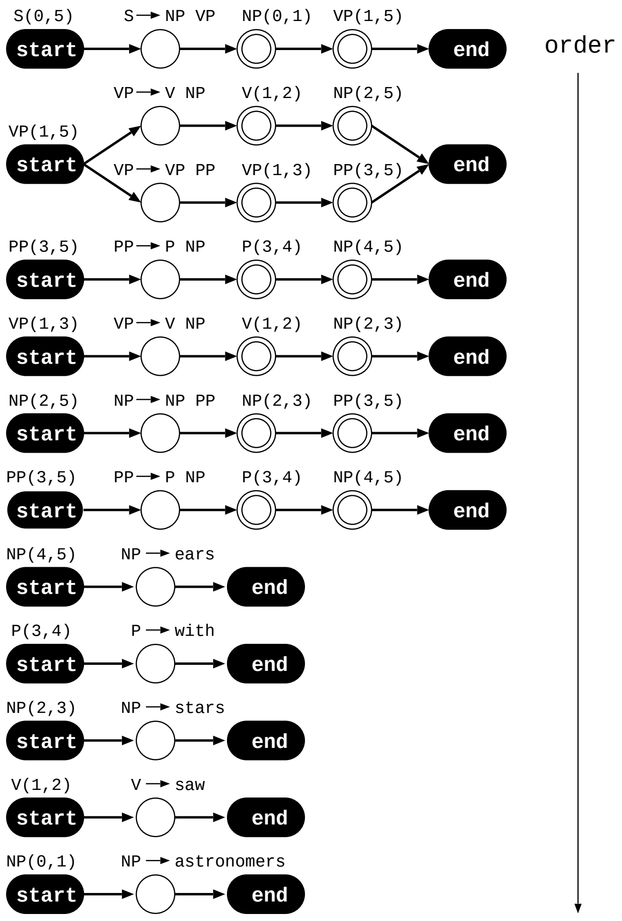

Figure 1: Parsing "astronomers saw stars with ears" and its support graph

[3]For simplicity, only PCFGs are considered here and we assume that there is neither the empty production rule nor non-terminal A such that $A \overset{+}{\Rightarrow} A$ in the grammar.

[4]This example is taken from (Manning and Schütze, 1999). We omit the grammar rules.

ordered set $\langle\tau_1,\ldots,\tau_M\rangle$ of disconnected subgraphs τ_k $(1 \le k \le M)$ which can be read off from the WFST for $\mathbf{w}^{(\ell)}$. Each τ_k is comprised of finitely many linear graphs ν of the form

$$\texttt{start-}N_1\texttt{-}\cdots\texttt{-}N_j\texttt{-end}.$$

`start` and `end` are respectively a fork node and a join node. The `start` node is labeled by an *index* of τ_k whose form depends on the type of parser we use. It is of the form $\mathrm{A}(i,j)$ for a CYK parser and is an item $[d' :_d A \rightarrow \zeta.\xi]$ for an Earley parser. Each internal node N_i $(1 \le i \le j)$ in ν is labeled by a *label* $\mathrm{label}(N_i)$ which is either a rule $A \leftarrow \zeta$ or an index of some other node in Δ_ℓ. We introduce a mapping $\tilde{\psi}_\ell(\cdot)$ that maps a disconnected subgraph τ_k to the set of sets of labels appearing in τ_k.

$$\begin{aligned}\mathcal{N}(\nu) &\stackrel{\mathrm{def}}{=} \{\mathrm{label}(N_i) \mid \text{internal node } N_i \in \nu\}\\ \tilde{\psi}_\ell(\tau_k) &\stackrel{\mathrm{def}}{=} \{\mathcal{N}(\nu) \mid \nu \text{ is a linear graph in } \tau_k\}\end{aligned}$$

The construction of a support graph for $\mathbf{w}^{(\ell)}$ proceeds as follows. By parsing $\mathbf{w}^{(\ell)}$ using our favorite parser, we obtain a WFST. From the WFST, we extract a partially ordered set of disconnected subgraphs $\{\tau_1,\ldots,\tau_M\}$ where each τ_i denotes a set of "similar" sub-derivations factored out from the whole derivation sequences. The partial ordering associated with these subgraphs is determined by the principle that τ_i precedes τ_j if τ_j is a sub-derivation of τ_i. Finally by topologically sorting the partially ordered set, we reach the totally ordered support graph $\Delta_\ell = \langle\tau_1,\ldots,\tau_M\rangle$ for the input sentence $\mathbf{w}^{(\ell)}$ where $\tau_1 = S(0,n_\ell)$ for a CYK parser and $\tau_1 = [n_\ell :_0 \rightarrow S.]$ for an Earley parser.

The actual extraction algorithm is somewhat optimized by merging the extraction of disconnected subgraphs with their sorting. A support graph extraction algorithm for a CYK parser is presented in (Sato et al., 2001) and one for an Earley parser is described in Section 6.

4 The graphical EM algorithm

After constructing support graphs $\{\Delta_\ell \mid 1 \le \ell \le T\}$, we run the graphical EM algorithm on them. It is driven by one main routine *gEM*() in Figure 2 that updates parameters[5] and two subroutines, *Get-Inside-Probs*() in Figure 3 which computes inside probabilities and *Get-Expectations*() in Figure 4 which computes outside probabilities together with the expected number of occurrences $\eta[A \leftarrow \zeta]$ of a rule $A \leftarrow \zeta \in R$ given the sentence.

[5] $P(\cdot \mid \theta)$ in *gEM*() stands for a distribution under the current parameter values θ.

In this section, we explain the graphical EM algorithm but for the ease of understanding and intuitiveness, we assume it runs on CYK type support graphs.[6]

Let $A(i,j)$ be an index labeling a disconnected subgraph τ_k in a support graph Δ_ℓ for $\mathbf{w}^{(\ell)}$. In each iteration of *gEM*(), *Get-Inside-Probs*() is called and recursively computes in a bottom-up manner each inside probability $P(A \stackrel{*}{\Rightarrow} \mathbf{w}^{(\ell)}[i,j])$ and stores it in array $\mathcal{P}[\ell, A(i,j)]$.[7] *Get-Inside-Probs*() also uses array $\mathcal{R}[\ell, A(i,j), E]$ to store probability $P(E)$ where $E = \{e_0,\ldots,e_M\} \in \tilde{\psi}(\tau_k)$. $P(E)$ is computed as a product $P(e_1)\cdots P(e_M)$ where $P(e) = \theta(A \leftarrow \zeta)$, i.e. the current parameter value of $A \leftarrow \zeta$ if node e is labeled by a rule $A \leftarrow \zeta \in R$, or $P(e) = \mathcal{P}[\ell, B(i',j')]$, i.e. the inside probability of $B(i',j')$ if e is labeled by an index $B(i',j')$. We see $\mathcal{P}[\ell, A(i,j)] = \sum_{E\in\tilde{\psi}(\tau_k)} P(E)$.

```
1:  procedure gEM() begin
2:    Initialize all parameters $\theta(A \rightarrow \zeta)$
3:      such that $P(\mathbf{w}^{(\ell)}|\theta) > 0$ for all $\ell = 1,\ldots,T$;
4:    Get-Inside-Probs();
5:    $\lambda^{(0)} := \sum_{\ell=1}^{T} \log \mathcal{P}[\ell, S(0,n_\ell)]$;
6:    repeat
7:      Get-Expectations();
8:      foreach $(A \rightarrow \zeta) \in R$ do
9:        $\theta(A \rightarrow \zeta) := \eta[A \rightarrow \zeta] / \sum_{\zeta'} \eta[A \rightarrow \zeta']$;
10:     $m$ += 1;
11:     Get-Inside-Probs();
12:     $\lambda^{(m)} := \sum_{\ell=1}^{T} \log \mathcal{P}[\ell, S(0,n_\ell)]$
13:   until $\lambda^{(m)} - \lambda^{(m-1)}$ is sufficiently small
14: end.
```

Figure 2: Main routine *gEM*

After computing all $\mathcal{P}[\ell, A(i,j)]$s, *Get-Expectations*() is called to compute each $\mathcal{Q}[\ell, A(i,j)]$, the outside probability $P(S \stackrel{*}{\Rightarrow} \mathbf{w}^{(\ell)}[0,i]\ A\ \mathbf{w}^{(\ell)}[j,n_\ell])$ in a top-down manner from $\tau_1 = \mathrm{S}(0,n_\ell)$ where $n_\ell = |\mathbf{w}^{(\ell)}|$, while incrementing the expected counts $\eta[A \leftarrow \zeta]$ of $A \leftarrow \zeta$ in a parse of $\mathbf{w}^{(\ell)}$. *gEM*() iterates to update parameters until an increase in the log likelihood of $\mathbf{w}^{(1)},\ldots,\mathbf{w}^{(T)}$ is less than a threshold, say 10^{-6}.

It is proved in (Kameya, 2000) that the graph-

[6] Adaptation by analogy to other types of parser is straightforward. In the case of an Earley parser for instance, $S(0,n_\ell)$ in *gEM*() is replaced by an initial item $[n_\ell :_0 \rightarrow S.]$.

[7] In the actual implementation, indecies are replaced by pointers to them.

```
procedure $Get\text{-}Expectations()$ begin
  foreach $(A \rightarrow \zeta) \in R$ do $\eta[A \rightarrow \zeta] := 0$;
  for $\ell := 1$ to $T$ do begin
    Put $\Delta_\ell = \langle \tau_1, \tau_2, \ldots, \tau_{|\Delta_\ell|} \rangle$;
    $\mathcal{Q}[\ell, \tau_1] := 1$;
    for $k := 2$ to $|\Delta_\ell|$ do $\mathcal{Q}[\ell, \tau_k] := 0$;
    for $k := 1$ to $|\Delta_\ell|$ do
      foreach $E \in \tilde{\psi}(\tau_k)$ do
        foreach $e \in E$ do
          if $e = (A \rightarrow \zeta)$ then $\eta[A \rightarrow \zeta] \mathrel{+}= \mathcal{Q}[\ell, \tau_k] \cdot \mathcal{R}[\ell, \tau_k, E] / \mathcal{P}[\ell, S(0, n_\ell)]$
          else if $\mathcal{P}[\ell, e] > 0$ then $\mathcal{Q}[\ell, e] \mathrel{+}= \mathcal{Q}[\ell, \tau_k] \cdot \mathcal{R}[\ell, \tau_k, E] / \mathcal{P}[\ell, e]$
  end  /* for $\ell$ */
end.
```

Figure 4: Subroutine *Get-Expectations*

```
procedure $Get\text{-}Inside\text{-}Probs()$ begin
  for $\ell := 1$ to $T$ do begin
    Put $\Delta_\ell = \langle \tau_1, \tau_2, \ldots, \tau_{|\Delta_\ell|} \rangle$;
    for $k := |\Delta_\ell|$ downto 1 do begin
      foreach $E \in \tilde{\psi}(\tau_k)$ do begin
        $\mathcal{R}[\ell, \tau_k, E] := 1$;
        foreach $e \in E$ do
          if $e = (A \rightarrow \zeta)$
          then $\mathcal{R}[\ell, \tau_k, E] \mathrel{*}= \theta(A \rightarrow \zeta)$
          else $\mathcal{R}[\ell, \tau_k, E] \mathrel{*}= \mathcal{P}[\ell, e]$;
      end;
      $\mathcal{P}[\ell, \tau_k] := \sum_{E \in \tilde{\psi}(\tau_k)} \mathcal{R}[\ell, \tau_k, E]$
    end  /* for $k$ */
  end  /* for $\ell$ */
end.
```

Figure 3: Subroutine *Get-Inside-Probs*

ical EM algorithm computes the same values (inside and outside probabilities, update values of parameters) as the I-O algorithm, hence the difference only lies in computational efficiency. Since time complexity of parsing (by a CYK parser and by an Earley parser) and that of one iteration for updating all parameters in one iteration by the graphical EM algorithm are all $O(N^3L^3)$ where N is the number of terminal symbols in a grammar and L the length of an input sentence,[8] we can say that the graphical EM algorithm is as efficient as the I-O algorithm. Likewise we can analyze time complexity by the graphical EM algorithm of various extensions of PCFGs. For example, parsing and parameter updating in one iteration for Pseudo PCSGs (Charniak and Carroll, 1994) take $O(N^4L^3)$ (Sato et al., 2001).

5 EM learning of PCFGs based on CYK charts

In this section, we experimentally compare the graphical EM algorithm with the I-O algorithm in terms of *time per iteration* (= time for updating parameters) by letting them learn PCFG parameters from corpora.[9]

Table 1 summarizes properties of ATR corpus (Uratani et al., 1994) and EDR corpus (Japan EDR, 1995) we used in the experiments. “min”, “ave.” and “max” in the column of sentence length respectively means the minimum, average and maximum length of sentences. Numbers in the column of grammar denote the number of rules contained in the used PCFG and the one with “(CF)” is that of the grammar translated into Chomsky normal form.

ATR corpus contains labeled 10,995 sentences. They are short and conversational Japanese sentences, and have a CFG grammar G_{atr} containing 860 rules which is not very ambiguous. G_{atr} is converted into Chomsky normal form G^*_{atr} containing 2,105 rules for the I-O algorithm. The corpus is divided into subgroups of similar length like $(L = 1, 2), (L = 3, 4), \ldots, (L = 25, 26)$, each containing randomly chosen 100 sentences.

For each subgroup of sentences, we applied the I-O algorithm to G^*_{atr} and also applied the graphical EM algorithm to CYK type support graphs

[8]It is easy to see that time complexity of the graphical EM algorithm in one iteration is linear in the size of a support graph which is $O(N^3L^3)$ (Kameya, 2000).

[9]Support graphs are CYK type and generated by a Tomita (Generalized LR) parser. All measurements were made on a 296MHz Sun UltraSPARC-II with Solaris 2.6 and the algorithms are implemented in C.

Corpus	size	sentence length	grammar	ambiguities at ave. length
ATR	10,995	min = 2, ave.= 9.97, max = 49	860/2,105(CF)	958 parses/sentence
EDR	9,900	min = 5, ave.= 20, max = 63	2,687/12,798(CF)	3.0×10^8 parses/sentence

Table 1: Data for ATR and EDR corpora

generated by $G_{\rm atr}$ and $G^*_{\rm atr}$ respectively. We plotted average updating time per iteration per sentence by varying $L = 2, 4, \ldots$[10] Figure 5 shows the result.

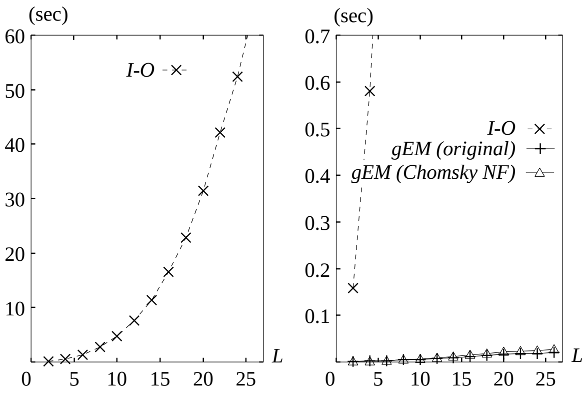

Figure 5: I-O vs. gEM (ATR)

In Figure 5, curves labeled "I-O" are drawn by the I-O algorithm[11] and those labeled "gEM(original)" and "gEM(Chomsky NF)" are drawn by the graphical EM, each corresponding to $G_{\rm atr}$ and to $G^*_{\rm atr}$ respectively.

The left graph is a learning curve drawn by the I-O algorithm. It clearly indicates cubic dependency of the I-O algorithm on sentence length. The right graph magnifies the y-axis of the left graph. Two curves, drawn by the graphical EM algorithm applied $G_{\rm atr}$ and by $G^*_{\rm atr}$ are added. They are very close. According to the learning data, at average length 10, the graphical EM algorithm applied to $G_{\rm atr}$ ran about 850 times faster than the I-O algorithm applied to $G^*_{\rm atr}$.

Such a big gap in learning speed is surprising but understandable because the I-O algorithm iterates the parsing of input sentences in every iteration while the graphical EM algorithm requires no parsing. Another factor to be considered is that ATR corpus contains short sentences and $G_{\rm atr}$ is not very ambiguous so that the resulting WFSTs are sparse. The sparseness of WFSTs implies the small size of the support graphs, which works advantageously to the graphical EM algorithm.

To confirm that the graphical EM algorithm can outperform the I-O algorithm even in the case of dense WFSTs, we repeated the same experiment as above using EDR corpus (Japan EDR, 1995) whose CFG grammar is much more ambiguous than the ones for ATR corpus. The corpus contains total 220,000 Japanese news articles but because it is under the process of re-annotation, we can use only part of it (randomly sampled 9,900 sentences) as a labeled corpus. As Table 1 indicates, it has a CFG grammar $G_{\rm edr}$ (2,687 rules) which is highly ambiguous, having 3.0×10^8 parses/sentence at length 20 and even 6.7×10^{19} at length 38. $G_{\rm edr}$ is converted into $G^*_{\rm edr}$ in Chomsky normal form that contains 12,798 rules.

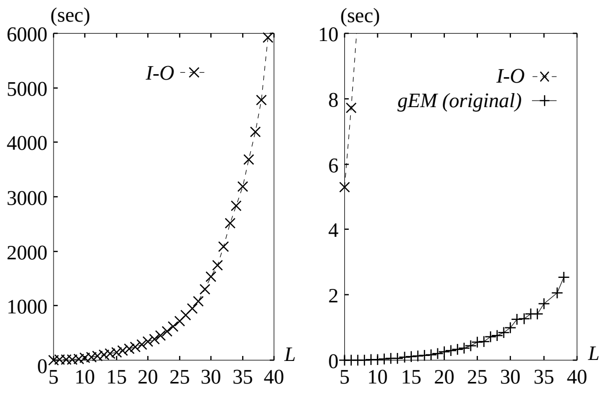

Figure 6: I-O vs. gEM (EDR)

Figure 6 shows the result of the same learning experiment with EDR corpus, i.e. the I-O algorithm applied to $G^*_{\rm edr}$ vs. the graphical EM algorithm applied $G_{\rm edr}$[12]

As was the case with ATR corpus, the I-O algorithm took so much time compared to the graphical EM algorithm that we have to separate their graphs. The left graph contains a curve plotted by the I-O algorithm and the right graph contains the magnified curve and a curve drawn by the graphical EM algorithm. This time at average sentence length 20, the graphical EM algorithm ran 1,300 times faster than the I-O algorithm per iteration.

[10]The stopping threshold was 10^{-6}.

[11]An x-axis is the length L corresponding to a subgroup of similar length and a y-axis is time per sentence taken by the EM algorithm to update all parameters in the grammar in one iteration.

[12]The plotted updating time is the average of 100 iterations for the graphical EM algorithm and the average of 20 iterations for the I-O algorithm, each per sentence.

The longer the sentence is, the larger the speed ratio becomes. Thus the speed ratio even widens compared to the case of ATR corpus. This can be explained by the mixed effects of time complexity $O(L^3)$ for the I-O algorithm and a slow increase in the size of support graphs for the graphical EM algorithm.

The comparisons made so far are based on EM learning time (per iteration). It should be noted however that the *total learning time* of the graphical EM algorithm consists of time for preprocessing (parsing and extracting support graphs) plus time for EM learning whereas the total learning time of the I-O algorithm merely consists of time for EM learning. Consequently making a comparison solely in terms of updating time thus gives a special favor to the graphical EM algorithm as it ignores time for preprocessing. We therefore made yet another comparison in terms of the total learning time using the entire ATR corpus. We do not give the details of the comparison here but the conclusion is that the graphical EM algorithm can still learn about 350 times faster than the I-O algorithm in this case (Sato et al., 2001).

It also must be considered that these conclusions could possibly depend on an implementation of the I-O algorithm because our implementation is faithful to (Baker, 1979) and naive. We thus conducted yet another learning experiment using an independent implementation by Mark Johnson of the I-O algorithm down-loadable from `http://www.cog.brown.edu/%7Emj/`. We measured time per iteration for the entire ATR corpus. His implementation turned out to be twice as fast as our naive implementation but still took 630 seconds per iteration whereas the graphical EM algorithm only took 0.661 second per iteration, giving a ratio of nearly 1,000 to 1.

(Sato et al., 2001) contains more detailed discussions of these experiments. It also contains learning experiments with a Pseudo PCSG and a lexicalized PCFG using the graphical EM algorithm on ATR corpus.

6 EM learning of PCFGs based on Earley charts

Stolcke's EM algorithm (Stolcke, 1995) runs on Earley charts. It incorporates parsing and factored computations of inside and outside probabilities. Since the redundancy in the I-O algorithm is eliminated, it is expected to run much faster than the I-O algorithm. In this section, we compare the graphical EM algorithm with Stolcke's EM algorithm in terms of updating time per iteration using ATR corpus and EDR corpus by conducting the same experiment as the previous section. [13]

6.1 Earley charts and Stolckes's EM algorithm

Stolcke's EM algorithm employs an Earley chart as data structure for EM learning. Let $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(T)}$ be T sampled sentences and I_ℓ an Earley chart for $\mathbf{w}^{(\ell)}$. I_ℓ contains items of the form $[d' : {}_dA \rightarrow \zeta.\xi]$ stating that ζ in a rule $A \rightarrow \zeta\xi$ already has spanned from position d to d' in $\mathbf{w}^{(\ell)}$. Items are constructed while parsing through a cycle of top-down prediction, scanning a word, and combining complete items as usual, but associated with each item are forward probabilities α, inner probabilities γ and outer probabilities β. The last two are generalizations of inside probabilities and outside probabilities respectively. For example the inner probability for the item $[d' : {}_dA \rightarrow \zeta.\xi]$ is the sum of probabilities of derivation paths from $[d : {}_dA \rightarrow .\zeta\xi]$ to $[d' : {}_dA \rightarrow \zeta.\xi]$. The outer probability has a more complex definition and omitted here (Stolcke, 1995). Let S be a start symbol. Then $[n_\ell : {}_0 \rightarrow S.]$ is the complete item signaling a successful parse where $n_\ell = |\mathbf{w}^{(\ell)}|$.

In every iteration, the algorithm updates parameter values by computing inner (inside) probabilities and outer (outside) probabilities. Note that inner probabilities are computed for all items in the chart but outer probabilities are computed only for items that lead to the complete item $[n_\ell : {}_0 \rightarrow S.]$.

Inner probabilities γ are updated by completion operation as follows. Let d' be the current dot position. If there are two items $[d' : {}_{d''}B \rightarrow \nu. \;\; [\gamma'']]$ and $[d'' : {}_dA \rightarrow \zeta.B\xi \;\; [\gamma]]$ in I_ℓ such that $d'' < d'$[14] where γ and γ'' are associated inner probabilities, replace an item of the form $[d' : {}_dA \rightarrow \zeta B.\xi \;\; [\gamma']]$ with $\gamma' {+}= \gamma \cdot \gamma''$ if it already exists in I_ℓ, i.e. the item has been constructed before. We implemented Stolcke's EM algorithm following (Stolcke, 1995) while supplementing implementation details.[15] So a queue is used to correctly combine new complete items and old items in the Earley chart. As a result, the computation of inside

[13]The experimental environment is the same as in Section 5, but the implementation language is changed to an object oriented programming language Ruby (see `http://www.ruby-lang.org/en/index.html` for details).

[14]Recall that our grammar is assumed not to contain the empty production rule.

[15]We have found that there are some subtleties concerning outer probability computation which we do not discuss here.

and outside probabilities using an Earley chart becomes dynamic in the sense that items to be combined are searched in every iteration in contrast to the graphical EM algorithm using a support graph that statically compiles the information on the correct combination of complete items into a totally ordered graph.

6.2 An extraction algorithm for Earley charts

Unlike Stolcke's EM algorithm, the graphical EM algorithm requires as input a support graph extracted from an Earley chart. The graph is obtained by running *Extract-Earley*() in Figure 7 on an Earley chart.

```
procedure $Extract\text{-}Earley()$ begin
  for $\ell := 1$ to $T$ do
    Initialize all $\tilde{\psi}_\ell(\cdot)$ to $\emptyset$ and all $Visited[\cdot]$ to NO;
    $ClearStack(U)$;
    $Visit\text{-}Earley(\ell, [n_\ell :_0 \rightarrow S.])$;
    for $k := 1$ to $|U|$ do $\tau_k := PopStack(U)$;
    $\Delta_\ell := \langle \tau_1, \tau_2, \ldots, \tau_{|U|} \rangle$
  end
end.
```

Figure 7: A support graph extraction algorithm for an Earley parser

Given an Earley chart I_ℓ for $\mathbf{w}^{(\ell)} = w_1^{(\ell)} \cdots w_{n_\ell}^{(\ell)}$ ($1 \leq \ell \leq T$), it calls a subroutine *Visit-Earley*() with a complete item $[n_\ell : {}_0 \rightarrow S.]$, and follows backtraces of item construction while pushing items into a stack U.

The generated graph, *Earley type support graph* $\Delta_\ell = \langle \tau_1, \tau_2, \ldots, \tau_{|U|} \rangle$ consists of totally ordered disconnected subgraphs such that each subgraph τ is comprised of one linear graph when it corresponds to prediction and scanning, or multiple linear graphs when it corresponds to completion of items. The map $\tilde{\psi}_\ell(\tau)$ giving the set of sets of labels appearing in τ is constructed in subroutine *Visit-Earley*() as follows.

$$\tilde{\psi}_\ell([d :{}_d B \rightarrow .\nu]) = \big\{\{B \rightarrow \nu\}\big\} \quad (1)$$

$$\tilde{\psi}_\ell([d' :{}_d A \rightarrow \zeta w_{d'}^{(\ell)} .\xi]) = \quad (2)$$
$$\big\{\{[(d'-1) :{}_d A \rightarrow \zeta . w_{d'}^{(\ell)} \xi]\}\big\}$$

$$\tilde{\psi}_\ell([d' :{}_d A \rightarrow \zeta B .\xi]) = \quad (3)$$
$$\Big\{\big\{[d'' :{}_d A \rightarrow \zeta . B\xi],\ [d' :{}_{d''} B \rightarrow \nu .]\big\}$$
$$\Big|\ d \leq d'' < d', [d' :{}_{d''} B \rightarrow \nu.] \in I_\ell \Big\}$$

```
procedure $Visit\text{-}Earley(\ell, [d' :{}_d A \rightarrow \zeta.\xi])$ begin
  Put $\tau = ([d' :{}_d A \rightarrow \zeta.\xi])$
  $Visited[\tau]$ := YES;
  if $\zeta = \varepsilon$ and $d' = d$ then $\tilde{\psi}_\ell(\tau) := \big\{\{A \rightarrow \xi\}\big\}$
          /* $[d :{}_d A \rightarrow .\xi]$ (Prediction) */
  else if $\zeta = \zeta' w_{d'}^{(\ell)}$ then begin
          /* $[d' :{}_d A \rightarrow \zeta' w_{d'}^{(\ell)} .\xi]$ (Scanning) */
    $\tilde{\psi}_\ell(\tau) := \big\{\{[(d'-1) :{}_d A \rightarrow \zeta' . w_{d'}^{(\ell)} \xi]\}\big\}$;
    if $Visited[[(d'-1) :{}_d A \rightarrow \zeta' . w_{d'}^{(\ell)} \xi]]$ = NO
    then
      $Visit\text{-}Earley(\ell, [(d'-1) :{}_d A \rightarrow \zeta' . w_{d'}^{(\ell)} \xi])$
    end
  else begin
    Put $\zeta = \zeta' B$;  /* $B$ is a non-terminal; */
            /* $[d' :{}_d A \rightarrow \zeta' B.\xi]$ (Completion) */
    foreach $d''$ such that $d \leq d'' < d'$ and
      $[d'' :{}_d A \rightarrow \zeta' . B\xi], [d' :{}_{d''} B \rightarrow \nu.] \in I_\ell$
      do begin
      Add to $\tilde{\psi}_\ell(\tau)$ a set $\big\{[d'' :{}_d A \rightarrow \zeta' . B\xi]$,
        $[d' :{}_{d''} B \rightarrow \nu.]\big\}$;
      if $Visited[[d'' :{}_d A \rightarrow \zeta' . B\xi]]$ = NO
      then $Visit\text{-}Earley(\ell, [d'' :{}_d A \rightarrow \zeta' . B\xi])$;
      if $Visited[[d' :{}_{d''} B \rightarrow \nu.]]$ = NO
      then $Visit\text{-}Earley(\ell, [d' :{}_{d''} B \rightarrow \nu.])$
    end
  end;
  $PushStack(\tau, U)$
end.
```

Figure 8: Subroutine *Visit-Earley*()

6.3 Comparing the two EM algorithms

To see the behavior of Stolcke's EM algorithm compared to the graphical EM algorithm, we conducted similar learning experiments as in Section 5 using ATR corpus.[16] We first parsed each subgroup of 100 sentences of the same length ($L = 1, 2, \ldots, 43, 44, 49$) chosen from the corpus using G_{atr} by an Earley parser and obtained Earley charts for each subgroup. When there are not an enough number of sentences of the same length, sentences are copied. Stolcke's EM algorithm is run directly on them. To remove an obvious redundancy of Stolcke's EM algorithm, inside probabilities were computed only for items that can be part of a parse tree. Also we extracted Earley type support graphs from the charts and ran the graphical EM algorithm on them. We plotted time per iteration per sentence varying L.

[16] We used a Unix machine with AMD Athlon 1.33GHz CPU, 768MB memory under FreeBSD 4.3 in the experiments.

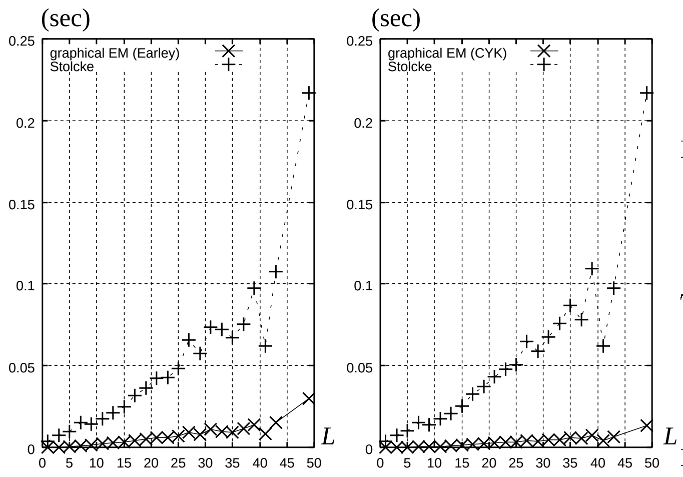

Figure 9: Stolcke's EM algorithm vs. gEM (ATR)

The left graph contains curves drawn by Stolcke's EM algorithm run on the complete item sets and the graphical EM algorithm run on the Earley type support graphs. Apparently the graphical EM algorithm runs faster than Stolcke's EM algorithm but the speed gap is much smaller as expected. Actually, at average length 10, the former runs 9.0 times faster than the latter and at other length larger than 10, the speed ratio varies from 8.3 to 6.7.[17] We also made a learning speed comparison using CYK type support graphs for the graphical EM algorithm the result of which is shown on the right graph. This time, the speed gap grows; at the average length 10, it is 43 and it is around 15 at other lengths larger than 10.

It seems that in spite of the elimination of the redundancy of computing useless inside probabilities on the side of Stolcke's EM algorithm (one of its two redundancies pointed out in Section 2), the graphical EM algorithm still outperforms Stolcke's EM algorithm by almost an order of magnitude or more, which suggests that the remaining redundancy (dynamic combination of complete items) considerable slows down the learning speed though we need more experiments to conclude.

References

J. K. Baker. 1979. Trainable grammars for speech recognition. In *Proceedings of Spring Conference of the Acoustical Society of America*, pages 547–550.

F. Beil, G. Carroll, D. Prescher, S. Riezler, and M. Rooth. 1999. Inside-Outside estimation of a lexicalized PCFG for German. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 269–276.

E. Charniak and G. Carroll. 1994. Context-sensitive statistics for improved grammatical language models. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pages 728–733.

T. Fujisaki, F. Jelinek, J. Cocke, E. Black, and T. Nishino. 1989. A probabilistic parsing method for sentence disambiguation. In *Proceedings of the 1st International Workshop on Parsing Technologies*, pages 85–94.

Ltd. Japan EDR. 1995. EDR electronic dictionary technical guide (2nd edition). Technical report, Japan Electronic Dictionary Research Institute, Ltd.

Y. Kameya. 2000. *Learning and Representation of Symbolic-Statistical Knowledge (in Japanese).* Ph. D. dissertation, Tokyo Institute of Technology.

K. Lari and S. J. Young. 1990. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56.

C. D. Manning and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing.* The MIT Press.

F. C. N. Pereira and Y. Schabes. 1992. Inside-Outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics (ACL'92)*, pages 128–135.

S. Riezler, J. Kuhn, D. Prescher, and M. Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measure and em training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, pages 480–487.

T. Sato, Y. Kameya, S. Abe, and K. Shirai. 2001. Fast EM learning of a family of PCFGs. Titech technical report (Dept. of CS) TR01-0006, Tokyo Institute of Technology.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

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

[17]The extra-time for generating support graphs (linear in the size of items in the chart) at length 10 costs 1.5 times of one iteration of the graphical EM algorithm.