CHAPTER 3

A SURVEY OF TECHNIQUES FOR FAST LEARNING OF PRODUCTION RULE PROBABILITIES

Several different techniques exist for learning the probability distributions associated with the production rules of a SCFG. Estimation of probabilities are typically performed using maximum likelihood (ML) optimization. The most popular ML techniques are the Inside-Outside (IO) algorithm (Baker, 1979; Lari and Young, 1990), that maximizes the likelihood of a dataset and the Viterbi Score (VS) algorithm (Ney, 1992), that maximizes the likelihood of the best derivation trees of a dataset. However, these techniques are far too complex to be of practical use in radar ES applications, which requires timely protection against threats. Several techniques for reducing the time complexities of IO and VS can be found in the literature (see Section 3.2).

In this thesis, a specific type of approach is considered, which is characterized by the use of chart parsers during pre-processing, to accelerate the iterative re-estimation of SCFG probabilities. More precisely, the techniques named Tree Scanning (TS), graphical EM (gEM), and HOLA will be studied. TS and gEM are EM techniques for ML optimization. TS corresponds to the extreme case that lists all the possible derivation trees of the sequences in the training dataset, and can lead to very fast execution for low-ambiguity grammars. However, memory requirements become an issue for high-ambiguity grammars. gEM requires more constant time complexity per iteration, and more moderate use of memory. Both TS and gEM produce the same results as IO. The original versions of these algorithms accelerate IO, and VS derivations of TS and gEM are introduced in this chapter. Finally, HOLA is a gradient descent technique for entropic optimization. These algorithms have been compared and discussed in (Latombe *et al.*, 2006c), (Latombe *et al.*, 2006a), and (Latombe *et al.*, 2006b).

This chapter first presents the classical ML approaches to approximating the probability distributions, and then describes the well-known IO and VS techniques. Then two fast alternatives to IO, namely the Tree Scanning, and the graphical EM techniques, along with their VS derivations, are reviewed. Finally, a gradient descent based technique called HOLA, for optimizing relative entropy is presented. The advantages and drawbacks of these techniques are discussed from an ES perspective.

3.1 Classical EM techniques based on ML approximation

In order to approximate a stochastic distribution defined over the training set, the problem of learning production rule probabilities of a SCFG from a set of sequences can be formulated as an optimization problem. Most popular techniques for optimizing or learning production rule probabilities are based on the EM algorithm, which guarantees that a local maximum is achieved. The objective function depends on the training set and is defined in terms of the probabilities of the rules. It uses growth transformations framework (Sanchez and Benedi, 1997), a special class of function (whose Eq. 3.2 belongs to), to re-estimate SCFG probabilities

Given a SCFG G_s, and any finite collection Ω of training sequences drawn from its language $Lg(G_s)$, with repetitions allowed, the maximum likelihood approach for learning the probabilities of the grammar consists of maximizing an objective function of the form (Nevado *et al.*, 2000):

$$P(\Omega, \Delta_\Omega | G_s) = \prod_{x \in \Omega} P(x, \Delta_x | G_s) \tag{3.1}$$

where $P(x, \Delta_x|G_s)$ is defined in Eq. 2.3. It can be noted that Eq. 3.1 coincides with the *likelihood* of the training sequences (Sanchez and Benedi, 1997) when Δ_x is identified with the set $\overline{\Delta}_x$ consisting of every possible derivation permitted by G_s and leading to $x \in \Omega$, and then $P(x|G_s) = P(x, \overline{\Delta}_x|G_s)$. It also coincides with the likelihood of the *best*

derivation of the sequence when Δ_x contains only the single most probable derivation $\hat{d}_x$ of $x \in \Omega$ (see Eq. 2.4). Both interpretations will be used in the subsequent analysis.

Optimization of Eq. 3.1 is normally implemented using an iterative Expectation-Maximization technique. At each iteration, the following function can be applied to re-estimate the production rule probabilities to approach a local maximum of Eq. 3.1 (Nevado *et al.*, 2000; Sanchez and Benedi, 1997):

$$\theta'(A \rightarrow \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s)}{\sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x | G_s)} \tag{3.2}$$

In Eq. 3.2, if Δ_x represents all the possible derivations $\overline{\Delta}_x$ for each sequence x in the training set, it corresponds to the IO algorithm, while if Δ_x represents only the best derivation $\hat{d}_x$ for each sequence x in the training set, it corresponds to the VS algorithm. It can also be noted that between the IO and VS algorithms, if Δ_x represents the k most probable derivations for each sequence in the training set, then Eq. 3.2 corresponds to the k-best derivation algorithm, or kVS (Nevado *et al.*, 2000; Sanchez and Benedi, 1997).

The rest of this section describes the well-known classical EM algorithms, called IO and VS, in more detail. Each one runs in an iterative manner, by using Eq. 3.2 to modify the probabilities of rules until a local optimum is achieved.

3.1.1 The Inside-Outside (IO) algorithm

The most popular algorithm optimizing the likelihood of the training dataset to re-estimate the probabilities of a SCFG is IO (Baker, 1979; Lari and Young, 1990, 1991), which is based on EM. This algorithm requires the grammar to be in Chomsky Normal Form (Lari and Young, 1990). A CFG is under CNF if its production rules are of the form $A \rightarrow BC$ (which will be called hereafter a transition rule) or $A \rightarrow a$ (which will be called hereafter an emission rule), where $\{A, B, C\}$ are non-terminals and a is a terminal. It is well known

that any CFG can be expressed in Chomsky Normal Form (Hopcroft *et al.*, 2001). As an example, the derivation trees shown in Fig. 3 do not represent a CNF grammar, while those shown in Fig. 8 do.

To re-estimate the probabilities, the IO algorithm computes during each iteration an inside and an outside probability in two passes. To be consistent with the classical notations associated with the CYK parser that will be used for the TS and gEM, all indexes will go from 0 to L. Since the production rules found by the parser will be of the form $A(i,j) \rightarrow B(i,k)C(k,j)$, where $A(i,j)$ indicates that the non-terminal A is at the origin of the subsequence $\{w_{i+1}...w_j\}$ of the parsed sequence $\{w_1...w_L\}$, the inside and outside probabilities of IO follow the same principle in their indexes. An iteration of the IO algorithm may be applied using the following steps:

a. Compute the inside probabilities: Given a training sequence $x = \{w_1, ..., w_L\} \in \Omega$, the inside algorithm computes a probability $\alpha(i-1, j|A) = P(A \Rightarrow w_i, ..., w_j)$ of a sub-tree starting at the non-terminal A and ending at $\{w_i, ..., w_j\}$ as shown in Fig. 6. It can be noted that $\alpha(0, L|Start)$ is the probability of the sequence to be generated by the grammar corresponding to all the possible derivation trees rooted at the initial non-terminal symbol, denoted by $Start$. The algorithm proceeds iteratively using the following recursive relations:

$$\begin{aligned} \alpha(i-1, i|A) &= \theta(A \rightarrow w_i) \\ \alpha(i,j|A) &= \sum_{B \in N} \sum_{C \in N} \sum_{k=i+1}^{j-1} \alpha(i,k|B)\alpha(k,j|C)\theta(A \rightarrow BC), \\ & \qquad \text{for } i < j-1 \end{aligned} \tag{3.3}$$

where N is the set of non-terminals of the grammar, A, B, and C are non-terminals, and w_i is the i^{th} word;

b. Compute the outside probabilities: Given a training sequence $\{w_1, ..., w_L\}$, the outside algorithm computes a probability $\beta(i, j|A) = P(Start \Rightarrow w_1...w_i A w_{j+1}...w_L)$ for all derivation trees containing the non-terminal A that generates the subsequence $\{w_1...w_i\}$ and $\{w_{j+1}...w_L\}$ outside of A (see Fig. 7). The computation of outside probabilities proceeds iteratively using the recursive relations:

$$\beta(0, L|Start) = 1$$

$$\beta(i, j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=0}^{i-1} \alpha(k, i|C)\beta(k, j|B)\theta(B \rightarrow CA) + \sum_{B \in N} \sum_{C \in N} \sum_{k=j+1}^{L} \alpha(j, k|C)\beta(i, k|B)\theta(B \rightarrow AC), \quad \text{for } i < j \qquad (3.4)$$

where L is defined as the size of a sequence.

c. Re-estimate the probabilities: IO uses the inside and outside probabilities to re-estimate the probabilities according to Eq. 3.2:

$$\theta'(A \rightarrow BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \le i < k < j \le L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}}$$

$$\theta'(A \rightarrow a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i|w_i = a} \beta(i-1,i|A)\theta(A \rightarrow a)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}} \qquad (3.5)$$

Note that α, β, and L depend on the training sequence x.

For reference, the routines for computing inside and outside probabilities, and for reestimating the probabilities are given in Algorithms 1 through 3, in which M_{nt} is the number of non-terminals of the grammar.

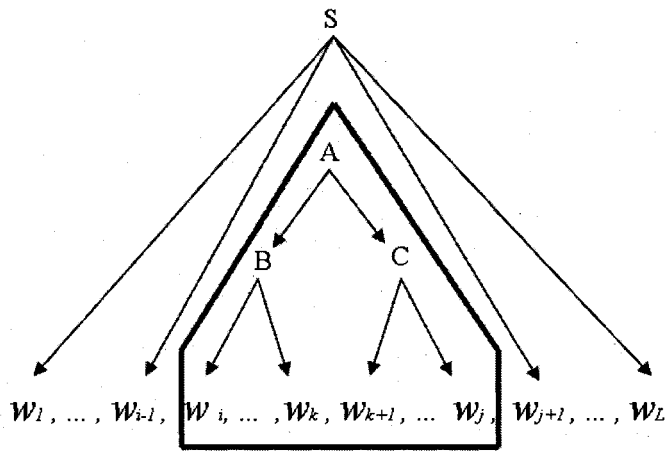

Figure 6 Branches of a SCFG that are relevant for computation of the inside probabilities of the IO algorithm.

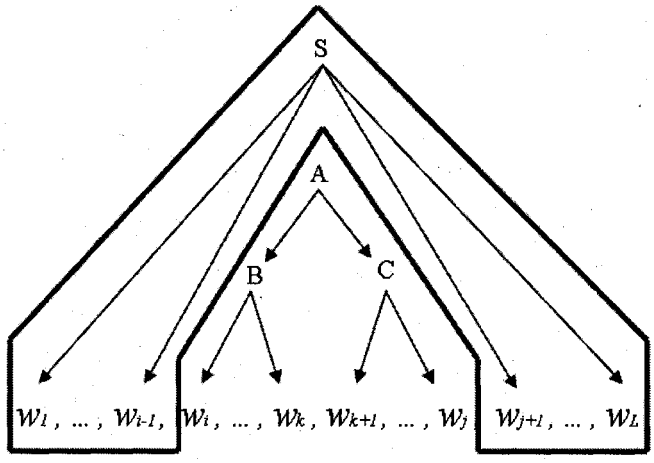

Figure 7 Branches of a SCFG that are relevant for computation of the outside probabilities of the IO algorithm.

Eq. 3.5 follows from Eq. 3.2 due to the following relations:

$$\sum_{d_x \in \bar{\Delta}_x} N(A \rightarrow BC, d_x) P(x, d_x | G_s) = \sum_{0 \leq i < k < j \leq L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)$$

Algorithm 1: `Inside(x)`

%%Initialization%%
for *i=1 to L* **do**
  for $A \in N$ **do**
    $\alpha(i-1, i|A) = \theta(A \rightarrow w_i)$;
%%Iteration%%
for *j=2 to L* **do**
  for *i=j-2 to 0* **do**
    for $A \in N$ **do**
      $\alpha(i,j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=i+1}^{j-1} \alpha(i,k|B)\alpha(k,j|C)\theta(A \rightarrow BC)$;

Algorithm 2: `Outside(x)`

%%Initialization%%
$\beta(0, L|Start) = 1$;
for $A \in N \backslash Start$ **do**
  $\beta(0, L|A) = 0$;
%%Iteration%%
for *i=0 to L-1* **do**
  for *j=L to i+1* **do**
    for $A \in N$ **do**
      $\beta(i,j|A) = \sum_{B \in N} \sum_{C \in N} \sum_{k=0}^{i-1} \alpha(k,i|C)\beta(k,j|B)\theta(B \rightarrow CA) +$
      $\sum_{B \in N} \sum_{C \in N} \sum_{k=j+1}^{L} \alpha(j,k|C)\beta(i,k|B)\theta(B \rightarrow AC)$;

$$\begin{aligned} \sum_{d_x \in \bar{\Delta}_x} N(A \rightarrow a, d_x) P(x, d_x|G_s) &= \sum_{i|w_i=a} \beta(i-1, i|A)\theta(A \rightarrow a) \\ \sum_{d_x \in \bar{\Delta}_x} N(A, d_x) P(x, d_x|G_s) &= \sum_{0 \le i < j \le L} \alpha(i,j|A)\beta(i,j|A) \end{aligned} \quad (3.6)$$

Consider the example shown in Fig. 8. This sequence corresponds to a phrase from an MFR of type Mercury, and will be used to train Mercury using IO. The Mercury Detection-Level Grammar given in Annex 3 is considered to have been initialized in a uniform way, which means that given a non-terminal A, every $\theta(A \rightarrow \lambda)$ is equal if $A \rightarrow \lambda$ appears

Algorithm 3: `Inside-Outside()`

while $loglikelihood(m) - loglikelihood(m-1) > \epsilon$ **do**
  for $x \in \Omega$ **do**
    $\alpha_x = \texttt{Inside}(x)$;
    $\beta_x = \texttt{Outside}(x)$;
  $loglikelihood(m) = \sum_{x \in \Omega} \alpha_x(0, L(x)|Start)$;
  for $A \in N$ **do**
    for $B \in N$ **do**
      for $C \in N$ **do**

$$\theta'(A \rightarrow BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < k < j \leq L} \alpha(i,k|B)\alpha(k,j|C)\beta(i,j|A)\theta(A \rightarrow BC)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < j \leq L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}};$$

    for $a \in V$ **do**

$$\theta'(A \rightarrow a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i|w_i = a} \beta(i-1,i|A)\theta(A \rightarrow a)}{\alpha(0,L|Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \leq i < j \leq L} \alpha(i,j|A)\beta(i,j|A)}{\alpha(0,L|Start)}};$$

  $m = m + 1$;

in the grammar definition. Then, the production probabilities appearing in Fig. 8 are the following:

$$\begin{aligned}
&\theta(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1 \\
&\theta(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1 \\
&\theta(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1 \\
&\theta(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667 \\
&\theta(\texttt{Na} \rightarrow Q6\ Q6) = 0.5 \\
&\theta(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408 \\
&\theta(Q6 \rightarrow W6\ W6) = 1 \\
&\theta(W6 \rightarrow 6) = 1 \\
&\theta(\epsilon \rightarrow 10) = 1
\end{aligned} \tag{3.7}$$

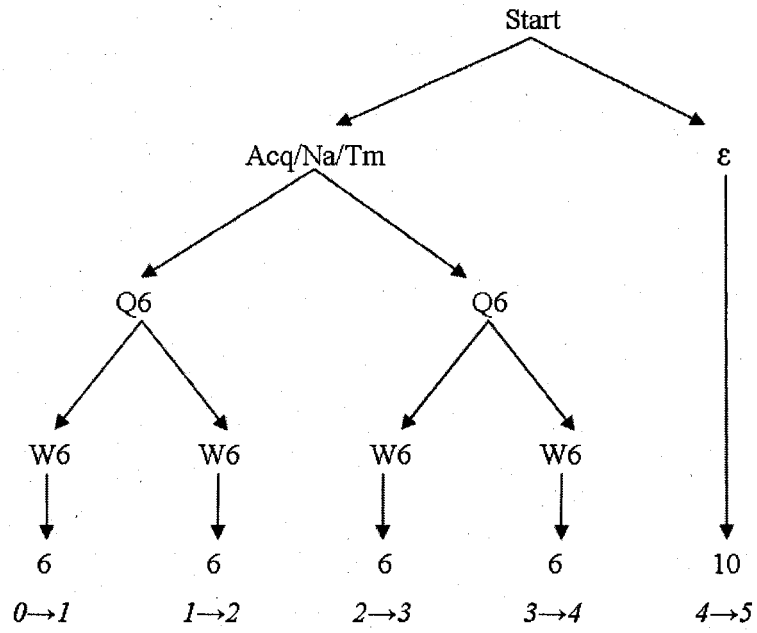

Figure 8 Example of a derivation tree from a short sequence emitted by an MFR.

Suppose that the quantities $\alpha(0,4|\texttt{Acq}) = 0.1667$, $\alpha(0,4|\texttt{Na}) = 0.15$, $\alpha(0,4|\texttt{Tm}) = 0.1408$ and $\alpha(4,5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.3, $\alpha(0,5|Start)$ will be computed as follows:

$$\begin{aligned}\alpha(0,5|Start) &= \alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon)\theta(Start \rightarrow \texttt{Acq}\ \epsilon) + \alpha(0,4|\texttt{Na})\alpha(4,5|\epsilon) \\ &\quad \theta(Start \rightarrow \texttt{Na}\ \epsilon) + \alpha(0,4|\texttt{Tm})\alpha(4,5|\epsilon)\theta(Start \rightarrow \texttt{Tm}\ \epsilon) \\ &= 0.1667 \cdot 1 \cdot 0.1 + 0.5 \cdot 1 \cdot 0.1 + 0.1408 \cdot 1 \cdot 0.1 = 0.08075 \quad (3.8)\end{aligned}$$

And according to Eq. 3.4, $\beta(0,4|\texttt{Acq})$ will be computed as follows:

$$\begin{aligned}\beta(0,4|\texttt{Acq}) &= \alpha(4,5|\epsilon)\beta(0,5|Start)\theta(Start \rightarrow \texttt{Acq}\ \epsilon) \\ &= 1 \cdot 1 \cdot 0.1 = 0.1 \quad (3.9)\end{aligned}$$

The reestimation formula of Eq. 3.5 for $\theta'(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned}\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\frac{1}{\alpha(0,5|Start)}\ \alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon)\beta(0,5|Start)\theta(Start \rightarrow \texttt{Acq}\ \epsilon)}{\frac{1}{\alpha(0,5|Start)}\ \alpha(0,5|Start)\beta(0,5|Start)} \\ &= \frac{\frac{1}{0.08075}\ 0.1667 \cdot 1 \cdot 1 \cdot 0.1}{\frac{1}{0.08075}\ 0.08075 \cdot 1} = 0.20644 \qquad (3.10)\end{aligned}$$

When IO estimates $\alpha(i,j|A)$ and $\beta(i,j|A)$, it passes through every possible combination of non-terminals $A \rightarrow BC$ as though each non-terminal could produce any pair of non-terminals, even if, according to the grammar, BC cannot be produced by A. Moreover, it considers that any non-terminal can have non-null inside and outside probabilities, whatever the subsequence. This will result in a time complexity per iteration of $O(M_{nt}^3 \cdot L^3)$, which increases exponentially with the number of non-terminals and the size of the sequence. On the other hand, it has the advantage of having a low memory complexity of $O(L^2 \cdot M_{nt})$.

It should be noted that for some applications, this algorithm can also be used for grammatical inference. For example, one can use IO with grammars that have different numbers of non-terminals, in which no probability is set to 0, and then select the best number of non-terminals (Lari and Young, 1990, 1991).

3.1.2 The Viterbi Score (VS) algorithm

While IO seeks to maximize the likelihood of a training set, the VS (Ney, 1992) algorithm, seeks to maximize the likelihood of the best derivations of a training set. Once a grammar is in CNF format, an iteration of the VS algorithm may be applied using the following steps:

a. Find the most likely derivation tree for the corresponding sequence: Let $\hat{\alpha}(i,j|A)$ represent the largest inside probability of any subtree rooted at a non-terminal

A, and generating the subsequence $w_{i+1}, ..., w_j$. Let $\psi(i, j|A)$ contain the rule $A(i, j) \rightarrow B(i, k)C(k, j)$ representing the highest vertex of the most probable subtree, including the non-terminals B and C and the word index k. For compact representation, let $\psi(i, j|A)$ refer only to the vector $[B, C, k]$, since it is enough to retrace the rule $A(i, j) \rightarrow B(i, k)C(k, j)$. These can be computed recursively using the following equations:

$$\begin{aligned}
&\hat{\alpha}(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \\
&\hat{\alpha}(i, j|A) = \max_{B,C} \max_{i<k<j} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}, \quad \text{for } i < j-1 \\
&\psi(i, j|A) = [B, C, k] = \text{argmax}_{B,C,k}\{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } i < j-1 \quad (3.11)
\end{aligned}$$

The derivation tree $\hat{d}_x$ can now be retraced by starting at $\psi(0, L|Start)$;

b. Count the number of times each rule appears in the derivation tree found from $\psi(0, L|Start)$;

c. Re-estimate the probabilities:

$$\text{If } \sum_{x\in\Omega} N(A, \hat{d}_x) \neq 0, \qquad \theta'(A \rightarrow \lambda) = \frac{\sum_{x\in\Omega} N(A \rightarrow \lambda, \hat{d}_x)}{\sum_{x\in\Omega} N(A, \hat{d}_x)} \qquad (3.12)$$

In the case of VS, Eq. 3.2 reduces to Eq. 3.12 since Δ_x contains only the best derivation tree $\hat{d}_x$.

For reference, the routines for finding the best derivation can be found in Algorithms 4 through 6. Algorithm 7 allows counting the frequency of the production rules and reestimating the probabilities.

Algorithm 4: `Maximum-Probability(x)`

%%Initialization%%
for *i=1 to L* **do**
  for $A \in N$ **do**
    $\hat{\alpha}(i-1, i|A) = \theta(A \rightarrow w_i)$;
%%Iteration%%
for *j=2 to L* **do**
  for *i=j-2 to 0* **do**
    for $A \in N$ **do**
      $\hat{\alpha}(i, j|A) = \max_{B,C \in N} \max_{i<k<j} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}$;
      $\psi(i, j|A) = \text{argmax}_{B,C,k} \{\theta(A \rightarrow BC)\hat{\alpha}(i, k|B)\hat{\alpha}(k, j|C)\}$;

Algorithm 5: `Retrace-path(x)`

$store_1^x = Start \rightarrow \psi(0, L|Start)(1)\psi(0, L|Start)(2)$;
$i = 0$;
$j = L$;
$k = \psi(0, L|Start)(3)$;
$B = \psi(0, L|Start)(1)$;
$C = \psi(0, L|Start)(2)$;
if $k - i > 1$ **then**
  `Add-store`$(\psi(i, k|B))$;
if $j - k > 1$ **then**
  `Add-store`$(\psi(k, j|C))$;

Algorithm 6: `Add-store` $(\psi(i, j|A))$

$store_{end+1}^x = A \rightarrow \psi(i, j|A)(1)\psi(i, j|A)(2)$;
$k = \psi(i, j|A)(3)$;
$B = \psi(i, j|A)(1)$;
$C = \psi(i, j|A)(2)$;
if $k - i > 1$ **then**
  `Add-store`$(\psi(i, k|B))$;
else
  $store_{end+1}^x = B \rightarrow w_k$;
if $j - k > 1$ **then**
  `Add-store`$(\psi(k, j|C))$;
else
  $store_{end+1}^x = C \rightarrow w_j$;

Algorithm 7: `Viterbi-Score()`

initialize $histo_t$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
initialize $histo_e$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
%%Histograms%%
for $x \in \Omega$ **do**
  for *i=1 to* $|store^x|$ **do**
    if $|store_i^x|$=*3* **then**
      $histo_t(store_i^x(1), store_i^x(2), store_i^x(3)) = histo_t(store_i^x(1), store_i^x(2), store_i^x(3)) + 1$;
    else
      $histo_e(store_i^x(1), store_i^x(2)) = histo_e(store_i^x(1), store_i^x(2)) + 1$;
%%Reestimation%%
for $A \in N$ **do**
  for $B \in N$ **do**
    for $C \in N$ **do**
      $\theta(A \rightarrow BC) = \frac{histo_t(A,B,C)}{\sum_{iD \in N} \sum_{E \in N} histo_t(A,D,E)}$;
  for $a \in V$ **do**
    $\theta(A \rightarrow a) = \frac{histo_e(A,a)}{\sum_{b \in V} histo_e(A,b)}$;

Consider the example of Fig. 8. According to the probabilities, the maximum likelihood derivation tree $\hat{d}_x$ is shown on Fig. 9, and the corresponding frequency of the rules is given in Eq. 3.13.

$$\begin{aligned} N(\texttt{Na} \rightarrow Q6\ Q6, \hat{d}_x) &= 1 \\ N(\texttt{Acq} \rightarrow Q6\ Q6, \hat{d}_x) &= 0 \\ N(\texttt{Tm} \rightarrow Q6\ Q6, \hat{d}_x) &= 0 \end{aligned} \tag{3.13}$$

Since `Na` can lead to other couples of non-terminals, $\theta(\texttt{Na} \rightarrow Q6\ Q6)$ has then to be normalized according to Eq. 3.12.

$$\theta'(\mathtt{Na} \rightarrow Q6\ Q6) = \frac{N(\mathtt{Na} \rightarrow Q6\ Q6, \hat{d}_x)}{\sum_\lambda N(\mathtt{Na} \rightarrow \lambda, \hat{d}_x)} \tag{3.14}$$

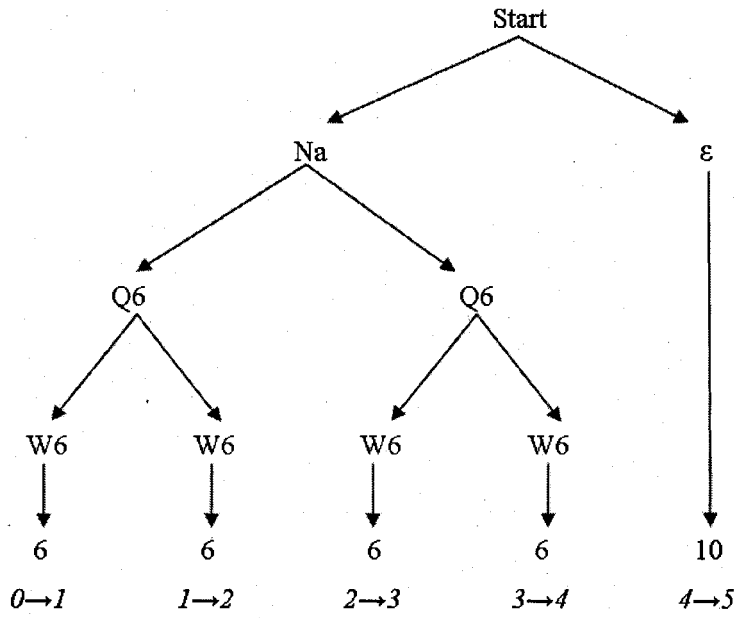

Figure 9 Best derivation tree for example in Fig. 8.

Both IO and VS have a worst-case time complexity per iteration that grows with $O(M_{nt}^3 L^3)$, and a memory complexity that grows with $O(L^2 M_{nt})$. However, it can be seen that, contrary to IO, VS requires only one pass to re-estimate the probabilities, and this gives a lower time complexity per iteration in practice. In addition VS is known to converge with fewer iterations than IO, even though the SCFGs are not, in general, as well learned (Sanchez and Benedi, 1999a).

Other alternatives to IO and VS include (1) the k-best derivation algorithm and (2) the VS algorithm with prior information. As they were not implemented for simulations, they are simply described in Annex 2.

3.2 Fast learning techniques based on ML approximation

Several alternatives have been proposed to accelerate the SCFG learning in different applications. Sakakibara (1990; 1993; 1994) uses Tree-Grammars, a technique to present the data such that it avoids passing through all the possibilities. Probability reestimation is then achieved using a generalization of HMM forward-backward algorithm and leads to a time complexity of $O(L^3 + M_{nt}^3 L)$. Kupiec (1992) uses an Hidden Markov Model (HMM) based representation of the grammar with trellis diagrams in order to compute the IO probabilities. For a same grammar, this algorithm has the same complexity as IO, but does not require the grammar to be in Chomsky Normal Form (CNF), which reduces the number of non-terminals M_{nt} and therefore results in a lower time complexity. Lucke (1994) proposes a BLI – the author does not define this acronym – algorithm in which the probabilities are approximated in a manner that is applicable to IO. It uses a stochastic parser, and perceives a derivation tree as a Bayesian network. Probabilities are re-estimated using two vectors called the evidential and causal support of a node. The approximations allow him to reduce IO time complexity from $O(M_{nt}^3 L^3)$ to $O(M_{nt}^2 L^3)$. Ito *et al.* (2001) reduces the time complexity from $O(M_{nt}^3)$ to $O(M_{nt}^2)$ by using restricted grammars, in which rules are of the form $A \rightarrow AB$ or $A \rightarrow BA$. The author explains that this kind of grammar can model many languages, including English. Finally, Chen and Chaudhari (2003) propose to use a prediction function before applying IO in order to detect some redundant operations. Modifying IO to avoid these operations allows reducing the time complexity per iteration by some unspecified amount.

In this thesis, a popular type of approach is considered to reduce the *time complexity per iteration* of IO. It involves pre-computing data structures such as support graphs and histograms, using tools like the Earley (Earley, 1970) or Cocke-Younger-Kasami (CYK) (Nijholt, 1991; Hopcroft *et al.*, 2001) parsers during the pre-processing phase. Pre-computation of data structures may then accelerate the iterative probability re-estimation process, since the blind combination of rules, where any non-terminal symbol could pro-

duce any combination of non-terminals, is avoided. All these techniques may in practice give lower time complexity, at the expense of an increased memory complexity. They are preferred in this work because MFR grammars are very simple (much more than natural language grammars for example). Thus, approaches that accelerate IO using the grammars structure rather than more general algorithmic improvement seems more appropriate. Fujisaki *et al.* (1989) were the first author to adapt this approach. They proposed using a CYK parser to find the derivations or the most probable derivations of the sentences, and then directly apply Eq. 3.2, corresponding either the Inside-Outside or the Viterbi algorithm. Based on this work, an algorithm called from now on Tree Scanning (TS) (Latombe *et al.*, 2006b), where all the possible derivation trees corresponding to a training set are computed in order to apply the basic reestimation equations, has been introduced. Two versions of the algorithms, one for IO,named TS(IO), and the other for VS, named TS(VS), have been proposed. If this algorithm is faster than IO in most of the applications, it has a time complexity of $O(M_{nt}^{L} \cdot L^3)$, and a memory complexity of $O(M_{nt}^{L} \cdot L)$ for a grammar of unbounded ambiguity. TS usually corresponds to the case where the most memory is sacrificed to accelerate time complexity.

Stolcke (1995) proposed an algorithm that computes the inside and outside probabilities of IO during the steps of an Earley parsing. However, this algorithm requires two passes – one for the inside probability and one for the outside probability. It has the same time complexity per iteration of $O(M_{nt}^{3} \cdot L^3)$ in the worst case, that is reduced to $O(M_{nt}^{3} \cdot L^2)$ for grammars of bounded ambiguity, and a memory complexity of $O(M_{nt}^{3} \cdot L^2)$.

Ra and Stockman (1999) introduced an extension of this last technique that computes both inside and probabilities in only one pass, using a special term that stores the weighted count of the rules appearing during the parsing, but increasing space complexity drastically to $O(M_{nt}^{6} \cdot L^2)$. Time complexity then becomes $O(\|r\|^2 L^3)$, where $\|r\|$ is the number of rules of the grammars. However, in the worst case, it is $O(M_{nt}^{6} L^3)$, which is more than

IO's. In this work, Stolcke and Ra's approaches are still too complex to be of practical use and were therefore not implemented.

More recently, Sato and Kameya (2001) initially used a chart parser to produce a special representation of the training data called support graphs, where only the combination of rules leading to the analyzed sequence are represented. Then, they run a new IO-like algorithm based on these support graphs called graphical EM (gEM). In literature, only an IO version of the algorithm is developed, a technique identified here in as gEM(IO). Therefore, a new version of gEM, named gEM(VS), that allows for applying VS to gEM, is proposed here. Their experiments show a five-fold reduction in time complexity per iteration on the ATR corpus (Uratani *et al.*, 1994), composed of 10995 short and conversational Japanese sentences. It will be shown that its time complexity per iteration and its memory complexity grow with $O(M_{nt}^3 \cdot L^3)$. It is worth noting that the techniques proposed by Fujisaki, Stolcke, Ra and Stockman, and Sato and Kameya compute the same values as IO and provide exactly the same results as IO.

Oates and Heeringa (2002) have introduced a heuristic incremental gradient descent algorithm called HOLA – the authors do not define this acronym – based on summary statistics. It uses a standard chart parser to compute the distributions of rules (the summary statistics) found after parsing the training database and after parsing a set of sequences produced by the grammar. Re-estimation of probabilities is performed using an approximation of the gradient descent (Annex 3.3). Unlike the other techniques, HOLA does not optimize explicitly the likelihood of a sequence (as with IO), but rather the relative entropy between these two distributions. It has the advantage of having very low time complexity per iteration and memory complexity of $O(M_{nt}^3)$.

The rest of this subsection presents a more detailed description of the TS, gEM, and HOLA algorithms.

3.2.1 The Tree Scanning (TS) Technique

The Tree Scanning (TS) algorithm consists in using a chart given by a classic Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) parser to find all the possible trees producing a sequence and then applying Eq. 3.2. In most cases it represents the limiting case in trading off time complexity for memory complexity. A possible pseudo-code to extract the trees after Earley parsing is given in Algorithms 8 and 9, and a possible pseudo-code to extract trees from a CYK chart is given in Algorithms 11 and 12.

Algorithm 8: `Earley-Extract()`

arrange *store*, the result from Earley parsing containing the rules (see Algorithm 28) by *stacks*, where each *stack* corresponds to the initial word of the subsequence stepped by the rule (see Annex 1 for more details on the Earley parser);
initialize the tools $dep1 = dep2 = [1]$;
`NewTree();`
while *stop=0* **do**
  foreach *stack* **do**
    `TreeCompletion();`
    `NewTree();`
    if *no element of dep1 appears in the stack as the producer of a rule* **then**
      $stop = 1$;

Algorithm 9: `NewTree()`

for *i=1 to* $|d_x|$ **do**
  for *m=1 to* $|dep2|$ **do**
    for *n=1 to* $|stack|$ **do**
      if $stack_n$ *is a transition rule* **then**
        $A \rightarrow BC = stack_n$;
      else
        $A \rightarrow a = stack_n$;
      if *A=deptemp(m)* **then**
        $d_x^{end+1} = d_x^i$;
        $dep1_{end+1} = dep1_i$;
$dept2 = dep1$;
remove all the redundant indexes from $dep2$;

Algorithm 10: `TreeCompletion()`

```
for i=1 to |d_x| do
    for m=1 to |dep1| do
        for n=1 to |stack| do
            if stack_n is a transition rule then
                A → BC = stack_n;
            else
                A → a = stack_n;
            if A=dep(m) then
                if stack_n is a transition rule then
                    add A → BC to d_x;
                    add B and C to dep1;
                else
                    add A → a to d_x;
```

Algorithm 11: `CYK-Extract()`

```
condition = 1;
stop = 0;
start = 1;
I = 0;
CYK-Trees(Start(0,L));
while stop=0 do
    start2=|d_x|+1;
    for l=start to |d_x| do
        A(i,j) → B(i,k)C(k,j) = chart_{d_x^{l,end}(4), d_x^{l,end}(5)}(I(l));
        CYK-Trees(B(i,k),l);
        CYK-Trees(C(k,j),l);
    if |d_x|=condition then
        stop = 1;
    condition=|d_x|;
    start = start2;
```

In both cases, the result is a set of stored rules that correspond to the derivation trees d_x in the algorithms. It is then easy to apply Eq. 3.2 according to the desired method (IO, VS or kVS).

Once the trees corresponding to the sequences of a training data set have been computed, an iteration of the TS algorithm may be applied using the following steps:

Algorithm 12: `CYK-Trees(A(i,j),numTree)`

count=0;
for *i=1 to* $|chart_{i,j}|$ **do**
  if *the rule corresponding to* $chart_{i,j}(i)$ *expands* A **then**
    $count = count + 1$;
    if *count=1* **then**
      $index = i$;
      if $chart_{i,j}(i)$ *is of the form* $B(k,m) \rightarrow C(k,l)D(l,m)$ **then**
        add $B \rightarrow C, D$ to $d_x^{numTree}$;
      else
        add $B \rightarrow Wi$ to $d_x^{numTree}$;
    else
      %%there is a new derivation: addition of a new tree%%
      $d_x^{end+1} = d_x^{numTree}$;
      if $chart_{i,j}(i)$ *is of the form* $B(k,m) \rightarrow C(k,l)D(l,m)$ **then**
        add $B \rightarrow C, D$ to d_x^{end};
      else
        add $B \rightarrow Wi$ to d_x^{end};
      $I(end+1) = i$;
    $resumption$=$|d_x|$;
    $B(k,m) \rightarrow C(k,l)D(l,m) = chart_{i,j}(index)$;
    if $d_x^{numTree}(resumption) = B \rightarrow CD$ **then**
      `CYK-Trees`$(C(k,l), numTree)$;
      `CYK-Trees`$(D(l,m), numTree)$;

a. Count the frequency of the rules and compute the probabilities of the trees:

   Use Eq. 2.2;

b. Re-estimate the probabilities: Use Eq. 3.2.

These two steps are performed using Algorithm 13 for the IO version of TS, named TS(IO)[1], and using Algorithm 14 for the VS version of TS, named TS(VS). The symbols

[1] Note that TS does not compute Inside and Outside probabilities. The notation just refers to the fact that all the derivation trees are considered during the re-estimation of the probabilities.

Algorithm 13: `Tree-Scanning(IO)`

while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  for *x=1 to* $|\Omega|$ **do**
    %%Histograms%%
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      for *j=1 to* $|d_x^i|$ **do**
        if $|d_x^i(j)|$*=3* **then**
          $histo\_t_{x,i}(d_x^i(j)) = histo\_t_{x,i}(d_x^i(j)) + 1$;
        else
          $histo\_e_{x,i}(d_x^i(j)) = histo\_e_{x,i}(d_x^i(j)) + 1$;
    %%Probability computation for each tree%%
    $p_{x,1to|d_x|} = 1$;
    for *i=1 to* $|d_x|$ **do**
      for *j=1 to* $|d_x^i|$ **do**
        $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;
    $prod\_t_x = \sum_i p_{x,i} \cdot histo\_t_{x,i}$;
    $prod\_e_x = \sum_i p_{x,i} \cdot histo\_e_{x,i}$;
    $ptotal_x = \sum_i p_{x,i}$;
  $cond^{(m)} = \sum_{x,i} p_{x,i}$;
  %%Reestimation%%
  $num\_t = \sum_x \frac{prod\_t_x}{ptotal_x}$;
  $num\_e = \sum_x \frac{prod\_e_x}{ptotal_x}$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  $m = m + 1$;

used in Alg. 13 can be linked to Eq. 3.2 as follows:

$$
\begin{aligned}
num_t(A,B,C) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow BC, d_x) P(x, d_x | G_s) \\
num_e(A,a) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow a, d_x) P(x, d_x | G_s)
\end{aligned}
\tag{3.15}
$$

Algorithm 14: `Tree-Scanning(VS)`

for *x=1 to* $|\Omega|$ **do**
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_{x,i} = p_{x,i} \cdot \theta(d_x^i(j))$;

while $cond^{(m)} - cond^{(m-1)} > \varepsilon$ **do**
  for *x=1 to* $|\Omega|$ **do**
    %%Histograms%%
    initialize $histo\_t_x$ as a null-valued matrix of size $M_{nt} \cdot M_{nt} \cdot M_{nt}$;
    initialize $histo\_e_x$ as a null-valued matrix of size $M_{nt} \cdot M_t$;
    for *i=1 to* $|d_x|$ **do**
      for $j = argmax_i\{p_{x,i}\}$ **do**
        if $|d_x^i(j)|$*=3* **then**
          $histo\_t_x(d_x^i(j)) = histo\_t_x(d_x^i(j)) + 1$;
        else
          $histo\_e_x(d_x^i(j)) = histo\_e_x(d_x^i(j)) + 1$;
  %%Reestimation%%
  $num\_t = \sum_x histo\_t_x$;
  $num\_e = \sum_x histo\_e_x$;
  for $i = 1$ *to* M_{nt} **do**
    $denom(i) = \sum_{j=1}^{M_{nt}} \sum_{k=1}^{M_{nt}} num\_t(i,j,k) + \sum_{j=1}^{M_t} num\_e(i,j)$;
  foreach *rule* $A \rightarrow BC$ *or* $A \rightarrow a$ **do**
    $\theta'(A \rightarrow BC) = \frac{num\_t(A,B,C)}{denom(A)}$;
    $\theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}$;
  %%Probabilities computation for each tree%%
  $p_{1to|d_x|} = 1$;
  for *i=1 to* $|d_x|$ **do**
    for *j=1 to* $|d_x^i|$ **do**
      $p_i = p_i \cdot \theta(d_x^i(j))$;
  $cond^{(m)} = \sum_x \max_i\{p_{x,i}\}$;
  $m = m + 1$;

$$
\begin{aligned}
denom(A) &= \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x | G_s) \\
\theta'(A \rightarrow BC) &= \frac{num\_t(A,B,C)}{denom(A)} \qquad \theta'(A \rightarrow a) = \frac{num\_e(A,a)}{denom(A)}
\end{aligned} \tag{3.16}
$$

Consider the example of Fig. 8. Three different trees lead to the sequence $x = 6\ 6\ 6\ 6\ 10$:

$$d_x^1 = [Start, \texttt{Acq}, \epsilon][\texttt{Acq}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

$$d_x^2 = [Start, \texttt{Na}, \epsilon][\texttt{Na}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

$$d_x^3 = [Start, \texttt{Tm}, \epsilon][\texttt{Tm}, Q6, Q6][Q6, W6, W6][W6, 6][W6, 6][Q6, W6, W6] \\ [W6, 6][W6, 6][\epsilon, 10]$$

where, for example, $[Start, \texttt{Tm}, \epsilon]$ represents the rule $Start \rightarrow \texttt{Tm}\ \epsilon$.

From these trees, it is easy to find $P(x, \Delta_x|G_s) = \alpha(0, 5|Start)$, using Eq. 2.2 and 2.3 with the production probabilities of Eq. 3.7:

$$\begin{aligned} P(x, \Delta_x|G_s) &= P(x, d_x^1|G_s) + P(x, d_x^2|G_s) + P(x, d_x^3|G_s) \\ &= \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} + \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} \\ &\quad + \prod_{A\rightarrow\lambda} \theta(A \rightarrow \lambda)^{N(A\rightarrow\lambda, d_x^1)} \\ &= 0.1 \cdot 0.1667 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 + 0.1 \cdot 0.5 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \\ &\quad + 0.1 \cdot 0.1408 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \\ &= 0.01667 + 0.05 + 0.01408 = 0.08075 \end{aligned} \tag{3.17}$$

For TS(IO), like for classical IO, $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ can be re-estimated:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{\sum_{x\in\Omega} \frac{1}{P(x,\Delta_x|G_s)} \sum_{d_x\in\Delta_x} N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x) P(x, d_x|G_s)}{\sum_{x\in\Omega} \frac{1}{P(x,\Delta_x|G_s)} \sum_{d_x\in\Delta_x} N(Start, d_x) P(x, d_x|G_s)} \tag{3.18}$$

where

$$\sum_{d_x \in \Delta_x} N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x) P(x, d_x^1 | G_s) = N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^1) P(x, d_x^1 | G_s) + N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^2) P(x, d_x^2 | G_s) + N(Start \rightarrow \texttt{Acq}\ \epsilon, d_x^3) P(x, d_x^3 | G_s) \tag{3.19}$$

and

$$\sum_{d_x \in \Delta_x} N(Start, d_x) P(x, d_x | G_s) = N(Start, d_x^1) P(x, d_x^1 | G_s) + N(Start, d_x^2) P(x, d_x^2 | G_s) + N(Start, d_x^3) P(x, d_x^3 | G_s) \tag{3.20}$$

which gives:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{\frac{1}{0.08075} \cdot (1 \cdot 0.01667 + 0 \cdot 0.05 + 0 \cdot 0.01408)}{\frac{1}{0.08075} \cdot (1 \cdot 0.01667 + 1 \cdot 0.05 + 1 \cdot 0.01408)} = 0.20644 \tag{3.21}$$

For TS(VS), like for classical VS, $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ can be re-estimated:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = \frac{N_{\hat{d_x}}(Start \rightarrow \texttt{Acq}\ \epsilon)}{N_{\hat{d_x}}(Start)} = \frac{N_{d_x^2}(Start \rightarrow \texttt{Acq}\ \epsilon)}{N_{d_x^2}(Start)} = \frac{0}{1} = 0 \tag{3.22}$$

This algorithm has the practical advantage of being very fast once the data has been pre-processed, when ambiguity is of low order of magnitude. Moreover the data from the pre-processing is very easy to store, as it does not require any particular order in the organization of rules. Although the memory requirements needed with the pre-processing are very low for low-ambiguity grammars, they become an issue for high-ambiguity grammars. It has the disadvantage, when ambiguity rises, of having a time complexity per iteration and a memory complexity that grow with $O(M_{nt}^{L} \cdot L^3)$ and $O(M_{nt}^{L} \cdot L)$, respectively.

3.2.2 The Graphical EM algorithm

During pre-processing, this algorithm creates a set of ordered support graphs from the chart of an Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) parser, to represent only the derivations that may possibly lead to a sequence. For the following, we will assume that a CYK parser has initially been used on the data (cf. Annex 1). From the resulting chart T, the algorithm creates support graphs, each one showing the possible productions from a non-terminal that generates the subsequence from w_{i+1} to w_j. Since the creation of the support graphs begins with $Start$, only production rules existing in the derivation trees leading to a given sequence will appear. The support graphs should be ordered such that a "father" is always before a "son", and the support graph generated by $Start$ is always first. Support graphs are extracted using routines `Extract-CYK` and `Visit-CYK` presented in Algorithms 15 and 16. Fig. 10 shows the CYK chart corresponding to the example in Fig. 8. Fig. 11 shows an example of support graphs created from the chart of Fig. 10, along with the corresponding notations.

During the iterative process gEM operates in a similar way to the IO algorithm. The main difference lies in the fact that gEM only passes by the transitions described in support graphs to re-estimate the probabilities. Once the support graphs have been computed,

each iteration of the gEM(IO) algorithm may be applied on the support graph of Fig. 11 using the following steps (Sato and Kameya, 2001).

Algorithm 15: `Extract-CYK`

for $l = 1to|\Omega|$ **do**
  Initialize all $\varphi(\tau)$ to $\emptyset$ and all $Visited[]$ to NO;
  ClearStack(U);
  `Visit-CYK`($l, Start, 0, L$);
  for *k=1 to* $|U|$ **do**
    τ_k := PopStack(U);
  $\delta_l := < \tau_1, ..., \tau_{|U|} >$;

Algorithm 16: `Visit-CYK` (l, A, i, j)

Put τ=A(i,j);
$Visited[\tau] := YES$;
if $j = i + 1$ **then**
  if $A(i,j) \rightarrow w_j^l \in chart(i,j)$ **then**
    add a set $\{A \rightarrow w_j\}$ to $\varphi\tau$;
else
  foreach $A(i,j) \rightarrow B(i,k)C(i,k) \in chart(i,j)$ **do**
    add to $\varphi(\tau)$ a set $A \rightarrow BC, B(i,k), C(k,j)$;
    if *Visited[B(i,k)]=NO* **then**
      `Visit-CYK`(l, B, i, k);
    if *Visited[C(k,j)]=NO* **then**
      `Visit-CYK`(l, C, k, j);
  PushStack(τ, U);

a. Compute the inside (α) and explanation (α_m) probabilities for each support graph in a bottom-up fashion:

Initialization:

$$\alpha(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \tag{3.23}$$

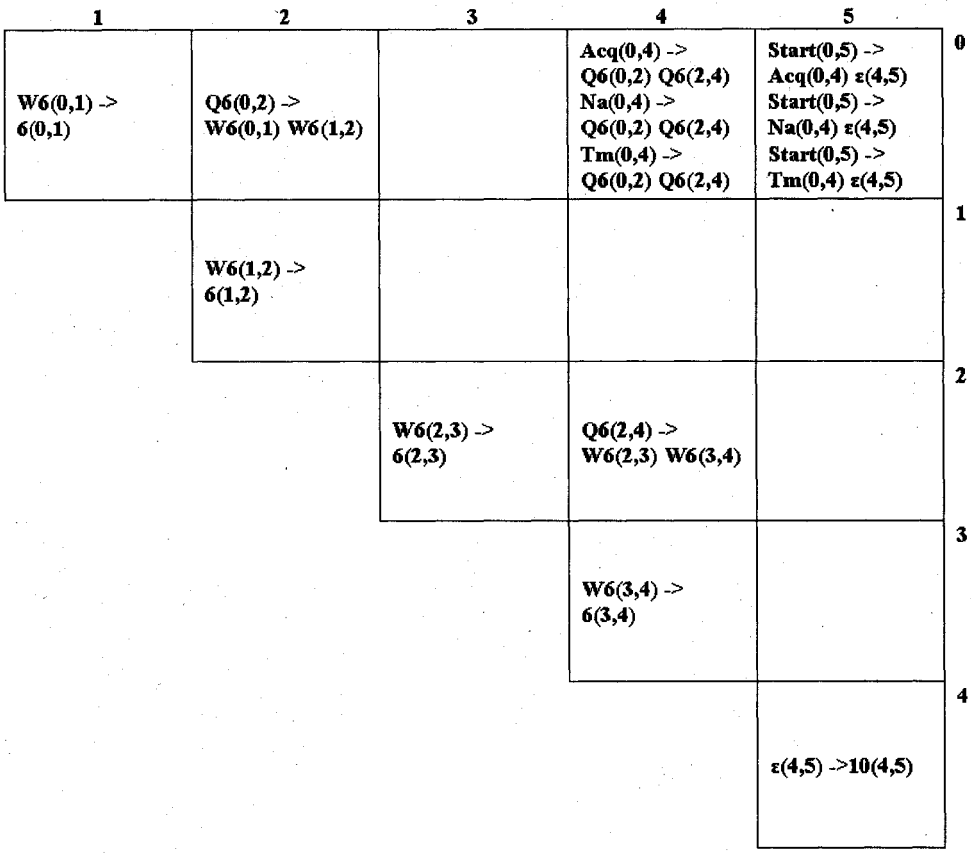

Figure 10 CYK tabular chart for example of Fig. 8.

Iterative process:

$$\begin{aligned} \alpha_m(i,j|A) &= \theta(A \rightarrow BC)\alpha(i,k|B)\alpha(k,j|C) \\ \alpha(i,j|A) &= \sum_m \alpha_m(i,j|A) \end{aligned} \tag{3.24}$$

where the summation extends over the branches of the support graph and the m^{th} branch represents a production rule of the form $A(i,j) \rightarrow B(i,k)C(k,j)$;

b. Compute the outside (β) probabilities and the balanced frequency of the rules (η) for each support graph in a top-down fashion:

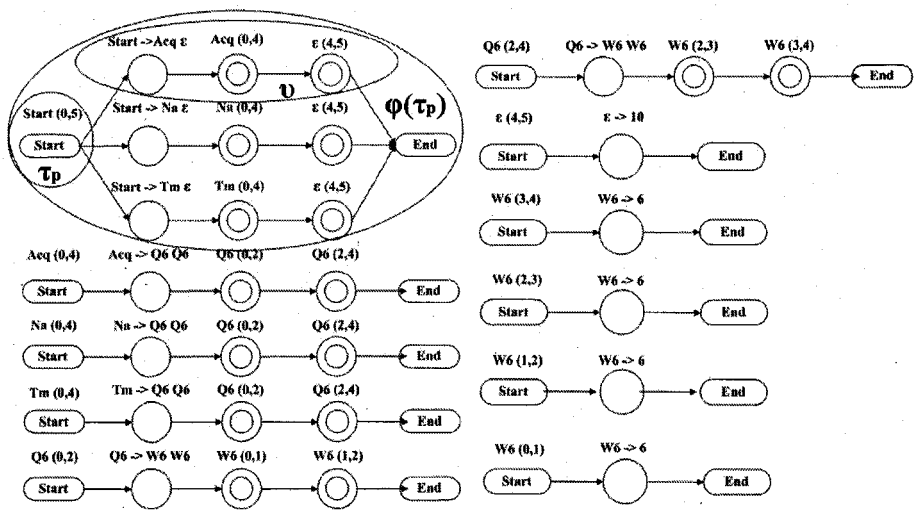

Figure 11 Support graphs for example of Fig. 8.

Initialization:

$$\begin{aligned} &\text{for } 0 \leq i < j \leq L,\ \beta(i,j|B) = 0, \quad \text{and} \quad \beta(0,L|Start) = 1 \\ &\text{and } \eta(A \rightarrow BC) = 0, \qquad A,B,C \in N \end{aligned} \tag{3.25}$$

Iterative process:

$$\begin{aligned} \beta(i,k|B) &\Leftarrow \beta(i,k|B) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(i,k|B)} \\ \eta(A \rightarrow BC) &\Leftarrow \eta(A \rightarrow BC) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(0,L|Start)} \\ \eta(A \rightarrow a) &\Leftarrow \eta(A \rightarrow a) + \frac{\beta(i,j|A)\alpha_m(i,j|A)}{\alpha(0,L|Start)} \end{aligned} \tag{3.26}$$

where L is the size of the sequence and the m^{th} branch of the support graph represents the production rule $A(i,j) \rightarrow B(i,k)C(k,j)$. It has been shown in (Sato *et al.*, 2001; Sato and Kameya, 2001) that these inside and outside probabilities cor-

respond to IO's, but are restrictions of Eq. 3.3 and 3.4 to the relevant contributing values from the support graphs;

c. Re-estimate the probabilities:

$$\theta'(A \rightarrow BC) = \frac{\eta(A \rightarrow BC)}{\sum_\lambda \eta(A \rightarrow \lambda)} \quad ; \quad \theta'(A \rightarrow a) = \frac{\eta(A \rightarrow a)}{\sum_\lambda \eta(A \rightarrow \lambda)} \tag{3.27}$$

For reference the inside and explanation probabilities are computed using the routine `Get-Inside-Probs(IO)` presented in Algorithm 18. Algorithm19 presents the routine `Get-Expectations(IO)` to compute the outside probabilities and η, and Algorithm 17 re-estimates the production probabilities.

The relation between the elements of these three steps and those of Eq. 3.2 can be expressed as follows.

$$\eta(A \rightarrow \lambda) = \sum_{x \in \Omega} \frac{1}{P(x, \Delta_x | G_s)} \sum_{d_x \in \Delta_x} N(A \rightarrow \lambda, d_x) P(x, d_x | G_s) \tag{3.28}$$

Algorithm 17: `Graphical-EM(IO)`

```
Get-Inside-Probs();
loglikelihood(0) = Σ_{x∈Ω} α_x(0, L(x)|Start);
while loglikelihood(m)-loglikelihood(m − 1) > ε do
    Get-Expectation();
    foreach (A → BC) ∈ R do
        θ'(A → λ) = η(A → λ)/ Σ_{λ'} η(A → λ');
    m = m + 1;
    Get-Inside-Probs();
    loglikelihood(m) = Σ_{x∈Ω} α_x(0, L(x)|Start);
```

Algorithm 18: `Get-Inside-Probs()`

for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  for $k = |\delta_l|$ *to 1* **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      $\alpha_E^l(\tau_k) = 1$;
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\alpha_E^l(\tau_k) = \alpha_E^l(\tau_k) \cdot \theta(A \rightarrow \lambda)$;
        else
          $\alpha_E^l(\tau_k) = \alpha_E^l(\tau_k) \cdot \alpha^l(e)$;
    $\alpha^l(\tau_k) = \sum_{E \in \varphi(\tau_k)} \alpha_E^l(\tau_k)$;

Algorithm 19: `Get-Expectations()`

foreach $(A \rightarrow \lambda) \in \alpha$ **do**
  $\eta(A \rightarrow \lambda) = 0$;
for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  $\beta^l(\tau_1) := 1$;
  for *k=2 to* $|\delta_l|$ **do**
    $\beta^l(\tau_k) := 0$;
  for *k=1 to* $|\delta_l|$ **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\eta(A \rightarrow \lambda) = \eta(A \rightarrow \lambda) + \beta^l(\tau_k) \cdot \alpha_E^l(\tau_k)/\alpha^l(0, L|Start)$;
        else
          if $\alpha^l(e) > 0$ **then**
            $\beta^l(e) = \beta^l(e) + \beta^l(\tau_k) \cdot \alpha_E^l(\tau_k)/\alpha_E^l(e)$;

Consider application of gEM to example of Fig. 8, using the production probabilities of Eq. 3.7. Suppose that the quantities $\alpha(0, 4|\texttt{Acq}) = 0.1667$, $\alpha(0, 4|\texttt{Na}) = 0.15$, $\alpha(0, 4|\texttt{Tm}) = 0.1408$ and $\alpha(4, 5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.24 $\alpha(0, 5|Start)$ will be computed as follows. Let $\alpha_1(0, 5|Start)$, $\alpha_2(0, 5|Start)$ and $\alpha_3(0, 5|Start)$ represent the explanation probabilities of the three branches in the first

support graph in Gig. 11. These can be computed as follows:

$$\begin{aligned}
\alpha_1(0,5|Start) &= \theta(Start \rightarrow \texttt{Acq}\ \epsilon)\alpha(0,4|\texttt{Acq})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.1667 \cdot 1 = 0.01667 \\
\alpha_2(0,5|Start) &= \theta(Start \rightarrow \texttt{Na}\ \epsilon)\alpha(0,4|\texttt{Na})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.5 \cdot 1 = 0.05 \\
\alpha_3(0,5|Start) &= \theta(Start \rightarrow \texttt{Tm}\ \epsilon)\alpha(0,4|\texttt{Tm})\alpha(4,5|\epsilon) \\
&= 0.1 \cdot 0.1408 \cdot 1 = 0.01408
\end{aligned} \tag{3.29}$$

Then $\alpha(0,5|Start)$ will be computed as in Eq. 3.30.

$$\begin{aligned}
\alpha(0,5|Start) &= \alpha_1(0,5|Start) + \alpha_2(0,5|Start) + \alpha_3(0,5|Start) \\
&= 0.01667 + 0.05 + 0.01408 \\
&= 0.08075
\end{aligned} \tag{3.30}$$

Moreover according to Eq. 3.26 $\eta(Start \rightarrow \texttt{Acq}\ \epsilon)$, $\eta(Start \rightarrow \texttt{Na}\ \epsilon)$ and $\eta(Start \rightarrow \texttt{Tm}\ \epsilon)$ will be computed as follows:

$$\begin{aligned}
\eta(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\beta(0,5|Start)\alpha_1(0,5|Start)}{\alpha(0,5|Start)} \\
&= \frac{1 \cdot 0.01667}{0.08075} = 0.20644
\end{aligned}$$

$$\begin{aligned}
\eta(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\beta(0,5|Start)\alpha_2(0,5|Start)}{\alpha(0,5|Start)} \\
&= \frac{1 \cdot 0.05}{0.08075} = 0.619195
\end{aligned}$$

$$\eta(Start \rightarrow \texttt{Tm}\ \epsilon) = \frac{\beta(0,5|Start)\alpha_3(0,5|Start)}{\alpha(0,5|Start)}$$

$$= \frac{1 \cdot 0.01408}{0.08075} = 0.174365 \tag{3.31}$$

The reestimation formula for $\theta(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned} \theta(Start \rightarrow \texttt{Acq}\ \epsilon) &= \frac{\eta(Start \rightarrow \texttt{Acq}\ \epsilon)}{\eta(Start \rightarrow \texttt{Na}\ \epsilon) + \eta(Start \rightarrow \texttt{Acq}\ \epsilon) + \eta(Start \rightarrow \texttt{Tm}\ \epsilon)} \\ &= \frac{0.20644}{0.20644 + 0.619195 + 0.174365} = 0.20644 \end{aligned} \tag{3.32}$$

It has been shown that the results produced by the graphical EM are the same as the results given by the IO algorithm (Sato *et al.*, 2001; Sato and Kameya, 2001). Actually, while the IO algorithm passes through all the possible combinations of a grammar to produce a sequence, the graphical EM algorithm only uses the combinations given by the support graphs. It is more efficient in most practical cases, although the worst case time complexity per iteration is equal to IO's. A greater memory complexity of $O(M_{nt}^3 \cdot L^2)$ is however needed to store the support graphs.

3.2.3 A VS version of gEM

Since VS has, in practice, a lower time complexity per iteration and usually converges faster that IO, a VS version of gEM(IO) has been proposed, leading to a new algorithm called gEM(VS) (Latombe *et al.*, 2006e). This algorithm modifies the support graphs in order to access the *best* derivation of the corresponding sequence, resulting in a set of one-branch support graphs. `Get-InsideProbs(VS)` of Algorithm 18 now computes the maximum probabilities instead of the inside probabilities, but still computes the explanation probabilities of the remaining parts of the support graphs. Fig. 12 shows a

possible result of the support graphs corresponding to gEM(VS) from the same example as in Fig. 11. Then, the routine `Get-Expectations(VS)` computes the outside probabilities and η based only on the new set of support graphs.

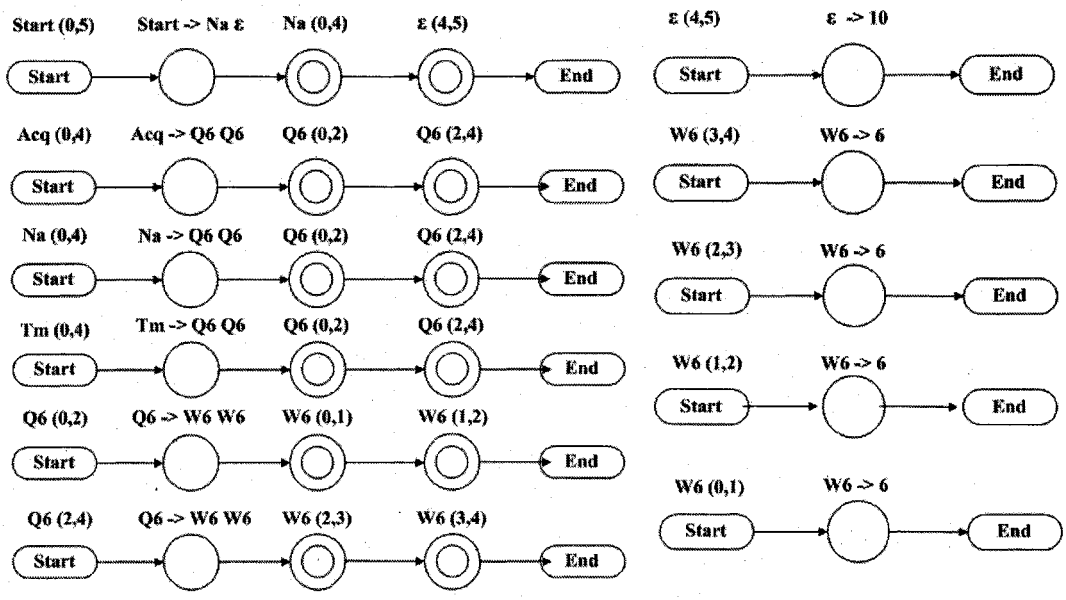

Figure 12 New support graphs for gEM(VS) based on Fig. 11.

Once the original support graphs have been precomputed, an iteration of the gEM(VS) algorithm may be applied on the support graph of Fig. 11 using the following steps:

a. Compute the maximum ($\hat{\alpha}$) and explanation ($\hat{\alpha}_m$) probabilities for each support graph in a bottom-up fashion:

Initialization:

$$\hat{\alpha}(i, i+1|A) = \theta(A \rightarrow w_{i+1}) \tag{3.33}$$

Iterative process:

$$\begin{aligned} \hat{\alpha_m}(i,j|A) &= \theta(A \rightarrow BC)\hat{\alpha}(i,k|B)\hat{\alpha}(k,j|C) \\ \hat{\alpha}(i,j|A) &= \max_m\{\hat{\alpha_m}(i,j|A)\} \end{aligned} \tag{3.34}$$

where the maximization extends over the branches of the support graphs and the m^{th} branch represents a production rule of the form $A(i,j) \rightarrow B(i,k)C(k,j)$;

b. Compute the outside ($\hat{\beta}$) probabilities and the balanced frequency of the rules ($\hat{\eta}$) for each support graph in a top-down fashion:

Initialization:

$$\begin{aligned} &\text{for } 0 \leq i < j \leq L,\ \hat{\beta}(i,j|B) = 0, \quad \text{and} \quad \hat{\beta}(0,L|Start) = 1 \\ &\qquad\qquad \text{and} \quad \hat{\eta}(A \rightarrow BC) = 0, \qquad A,B,C \in N \end{aligned} \tag{3.35}$$

Iterative process:

$$\text{If } \hat{\beta}(i,k|A) \neq 0: \qquad \begin{aligned} \hat{\beta}(i,k|B) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(i,k|B)} \\ \hat{\eta}(A \rightarrow BC) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(0,L|Start)} \\ \hat{\eta}(A \rightarrow a) &= \frac{\hat{\beta}(i,j|A)\hat{\alpha_m}(i,j|A)}{\hat{\alpha}(0,L|Start)} \end{aligned} \tag{3.36}$$

Here, m identifies the most probable branch of the support graph as determined by Eq. 3.34, and corresponds to the production rule $A(i,j) \rightarrow B(i,k)C(k,j)$. L is still the size of the sequence. In the computation of $\hat{\eta}$, the normalization is always performed with respect to $\hat{\alpha}(0,L|Start)$, whatever the corresponding rule;

c. Reestimate the probabilities:

$$\theta'(A \rightarrow BC) = \frac{\hat{\eta}(A \rightarrow BC)}{\sum_{\lambda} \hat{\eta}(A \rightarrow \lambda)} \quad ; \quad \theta'(A \rightarrow a) = \frac{\hat{\eta}(A \rightarrow a)}{\sum_{\lambda} \hat{\eta}(A \rightarrow \lambda)} \tag{3.37}$$

These steps are performed using Algorithms 20 through 22.

The relation between the elements of these steps and those of Eq. 3.2 can be expressed as follows.

$$\hat{\eta}(A \rightarrow \lambda) = \sum_{x \in \Omega} N(A \rightarrow \lambda, \hat{d}_x) \tag{3.38}$$

Algorithm 20: `Graphical-EM(VS)`

```
Get-Inside-Probs-VS();
loglikelihood(0) = Σ_{x∈|Ω|} α_x(0, L(x)|Start);
while loglikelihood(m)-loglikelihood(m − 1) > ε do
    Get-Expectation-VS();
    foreach (A → BC) ∈ R do
        θ'(A → λ) = η̂(A → λ)/ Σ_{λ'} η̂(A → λ');
    m = m + 1;
    Get-Inside-Probs-VS();
    loglikelihood(m) = Σ_{x∈|Ω|} α_x(0, L(x)|Start);
```

The support graphs of the example of Section 3.2.2 obtained by using the Viterbi version of the algorithm are given in Fig. 12. Suppose that the production rule probabilities have been initialized like in Eq. 3.7, and that the quantities $\hat{\alpha}(0, 4|\texttt{Acq}) = 0.1667$, $\hat{\alpha}(0, 4|\texttt{Na}) = 0.5$, $\hat{\alpha}(0, 4|\texttt{Tm}) = 0.1408$ and $\hat{\alpha}(4, 5|\epsilon) = 1$ have already been computed. Then, according to Eq. 3.34 $\hat{\alpha}(0, 5|Start)$ will be computed as follows:

$$\begin{aligned}
\hat{\alpha}(0, 5|Start) &= \max_{m}\{\hat{\alpha}_m(0, 5|Start)\} \quad . \\
&= \max\{\theta(Start \rightarrow Acq\epsilon)\hat{\alpha}(0, 4|\texttt{Acq})\hat{\alpha}(4, 5|\epsilon), \theta(Start \rightarrow Na\epsilon)\hat{\alpha}(0, 4|\texttt{Na}))\hat{\alpha}(4, 5 \\
&= \theta(Start \rightarrow Na\epsilon)\hat{\alpha}(0, 4|\texttt{Na}))\hat{\alpha}(4, 5|\epsilon) \\
&= 0.05
\end{aligned}$$

Algorithm 21: `Get-Inside-Probs-VS()`

for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  for $k = |\delta_l|$ *to 1* **do**
    foreach $E \in \varphi(\tau_k)$ **do**
      $\hat{\alpha}^l_E(\tau_k) = 1$;
      foreach $e \in E$ **do**
        if $e = (A \rightarrow \lambda)$ **then**
          $\hat{\alpha}^l_E(\tau_k) = \hat{\alpha}^l_E(\tau_k) \cdot \theta(A \rightarrow \lambda)$;
        else
          $\hat{\alpha}^l_E(\tau_k) = \hat{\alpha}^l_E(\tau_k) \cdot \alpha[l, e]$;
    $\hat{\alpha}^l(\tau_k) = \max_{E \in \varphi(\tau_k)}\{\hat{\alpha}^l_E(\tau_k)\}$;
    $E_{max} = \text{argmax}_{E \in \varphi(\tau_k)}\{\hat{\alpha}^l_E(\tau_k)\}$;
    $\psi^l_{viterbi}(\tau_k) = \varphi^l(\tau_k)(E_{max})$;

Algorithm 22: `Get-Expectations-VS()`

foreach $(A \rightarrow \lambda) \in R$ **do**
  $\eta[A \rightarrow \zeta] = 0$;
for *l=1 to* $|\Omega|$ **do**
  Put $\delta_l =< \tau_1, ..., \tau_{|\delta_l|} >$;
  $\hat{\beta}^l(\tau_1) := 1$;
  for *k=2 to* $|\delta_l|$ **do**
    $\hat{\beta}^l(\tau_k) := 0$;
  for *k=1 to* $|\delta_l|$ **do**
    foreach $e \in \varphi_{viterbi}(\tau_k)$ **do**
      if $e = (A \rightarrow \lambda)$ **then**
        $\hat{\eta}(A \rightarrow \lambda) = \hat{\eta}(A \rightarrow \lambda) + \hat{\beta}^l(\tau_k) \cdot \hat{\alpha}^l_E(\tau_k)/\hat{\alpha}^l(S(0, n_l))$;
      else
        if $\hat{\alpha}^l(e) > 0$ **then**
          $\hat{\beta}^l(e) = \hat{\beta}^l(e) + \hat{\beta}^l(\tau_k) \cdot \hat{\alpha}^l_E(\tau_k)/\hat{\alpha}^l(e)$;

Moreover according to Eq. 3.36 $\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon)$, $\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon)$ and $\hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon)$ will be computed as follows:

$$\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon) = 0$$

$$\begin{aligned}\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\hat{\beta}(0,5|Start)\hat{\alpha}_2(0,5|Start)}{\hat{\alpha}(0,5|Start)} = \frac{1 \cdot 0.05}{0.05} = 1 \\ \hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon) &= 0 \end{aligned} \tag{3.40}$$

The reestimation formula for $\theta'(Start \rightarrow \texttt{Acq}\ \epsilon)$ gives:

$$\begin{aligned}\theta'(Start \rightarrow \texttt{Na}\ \epsilon) &= \frac{\hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon)}{\hat{\eta}(Start \rightarrow \texttt{Acq}\ \epsilon) + \hat{\eta}(Start \rightarrow \texttt{Na}\ \epsilon) + \hat{\eta}(Start \rightarrow \texttt{Tm}\ \epsilon)} \\ &= \frac{1}{0+1+0} = 1 \end{aligned} \tag{3.41}$$

And the probabilities have to be normalized as in Eq. 3.37.

After the first step of the iterative process, all the support graphs are kept. Indeed, as `Get-Inside-Probs(VS)` runs in a bottom-up fashion, support graphs coming from a deleted branch located higher in the support graphs will be present at the end of the routine. As `Get-Expectations(VS)` works only on the new support graph in a top-down fashion, and thanks to the initialization of $\hat{\beta}$, the useless support graphs will not have effect on the computation of $\hat{\eta}$. Indeed $\hat{\eta}$ is computed by considering only the best derivation tree. Since the second pass must be performed on these one-branch support graphs, gEM(VS) requires more memory than gEM(IO) for storage. However, both gEM(VS) and gEM(IO) have time and memory complexities that grow with $O(M_{nt}^3 \cdot L^3)$ and $O(M_{nt}^3 \cdot L^2)$.

3.3 Fast gradient descent based on relative entropy – HOLA

Oates and Heeringa (2002) present a heuristic on-line algorithm called HOLA based on summary statistics. Unlike IO, VS, TS or gEM, HOLA does not optimize the likelihood of the training dataset in a EM fashion to re-compute production rule probabilities. Instead, it re-estimates the probabilities of a grammar by using an approximation of the gradient descent. It requires pre-processing prior to reestimation of the probabilities. A routine

computes summary statistics for the training database. A standard Earley (Earley, 1970) or CYK (Nijholt, 1991; Hopcroft *et al.*, 2001) chart parser is used to count how many times a particular rule is used in the production of the sequences of the training database. For a grammar under CNF the amount of data needed is two matrices, one for the transition rules and a second for the emission rules. These matrices are stored, and can then be updated if new data is added. The values of the frequencies of the rules are then normalized according to Eq. 3.42:

$$N'(A \rightarrow \lambda).o = \frac{N(A \rightarrow \lambda).o}{N(A).o} \tag{3.42}$$

where $N(A \rightarrow \lambda).o$ represents the frequency of appearance of the rule $A \rightarrow \lambda$ when parsing the training dataset, and $N(A).o$ represents the frequency of A being the origin of any production rule. This allows a comparison of counts with different numbers of sequences. For example of Fig. 10, the $N().o$ will store the following values:

$$\begin{aligned}
&N(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1 \quad && N(Start \rightarrow \texttt{Na}\ \epsilon).o = 1 \\
&N(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1 \quad && N(\texttt{Acq} \rightarrow Q6\ Q6).o = 1 \\
&N(\texttt{Na} \rightarrow Q6\ Q6).o = 1 \quad && N(\texttt{Tm} \rightarrow Q6\ Q6).o = 1 \\
&N(Q6 \rightarrow W6\ W6).o = 2 \quad && N(W6 \rightarrow 6).o = 4 \\
&N(\epsilon \rightarrow 10).o = 1
\end{aligned} \tag{3.43}$$

Once the histograms have been computed, an iteration of the HOLA algorithm may be applied using the following steps, as presented in Algorithm 25:

a. Generate sequences from the SCFG G_s: Starting from the $Start$ symbol, randomly generate a particular set of derivation trees that lead to a corresponding set of sequences, using the probabilities of G_s. In other words, the sequences depend on the current distribution of the probabilities of G_s;

b. Compute and normalize the corresponding frequency histograms: Parse the generated sequences and count the total number of times $N(A \rightarrow \lambda).s$ that the production rule $A \rightarrow \lambda$ occurs in the complete set of CYK charts. Then normalize to compute $N'(A \rightarrow \lambda).s$;

c. Re-estimate the probabilities:

$$\theta'(A \rightarrow \lambda) = \theta(A \rightarrow \lambda)(1 + \chi(N'(A \rightarrow \lambda).o - N'(A \rightarrow \lambda).s)) \tag{3.44}$$

Note that $\theta'(A \rightarrow \lambda)$ must be subsequently normalized to remain consistent.

The pseudo-codes of the algorithms are given below in Algorithms 23 through 25.

Algorithm 23: `HOLA()`

```
load HOLA-Count-Storage-Normalized;
while criterion is not reached do
    HOLA-Iteration;
```

Algorithm 24: `HOLA-Count()`

```
derivation = parse each sequence of the database;
foreach d ∈ derivation do
    foreach rule r in the grammar do
        if r = d then
            N(r).o = N(r).o + 1;
save HOLA-Count-Storage-Raw;
{N'(r).o}=normalize {N(r).o};
save HOLA-Count-Storage-Normalized;
```

Algorithm 25: `HOLA-Iteration()`

generate a set of sequences according to the grammar;
derivation = parse each generated sequence;
foreach $d \in$ *derivation* **do**
    foreach *rule* r *in the grammar* **do**
        if $r = d$ **then**
            $N(r).s = N(r).s + 1$;
$\{N'(r).s\}$=normalize $\{N(r).s\}$;
foreach *rule* r **do**
    $\theta(r) = \theta(r) \cdot (1 + \chi \cdot (N'(r).o - N'(r).s))$;
    if $\theta(r) \leq 0$ **then**
        $\theta(r) = min$;

It can be seen that three parameters must be set with HOLA – the number of sequences to randomly generate at each iteration, the learning rate χ in the reestimation formula, and the stopping criterion. Parameter values depend on the application, and will vary according to the average size of the sequences, the size of the grammar, etc. As the purpose of HOLA is to have the number $N'(A \rightarrow \lambda).s$ tend toward $N'(A \rightarrow \lambda).o$ for every rule $A \rightarrow \lambda$, a possible stopping criterion consists in making the relative entropy converge on an independent validation set. The relative entropy of two distributions p and q is given by:

$$H = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{3.45}$$

In this case, for a grammar under CNF, p and q are the sample and the observed distribution of the frequency of a non-terminal producing a combination of two non-terminals or producing a terminal. The sum of the relative entropies over all the non-terminals becomes:

$$HH = \sum_{A \rightarrow \lambda} N'(A \rightarrow \lambda).s \log \frac{N'(A \rightarrow \lambda).s}{N'(A \rightarrow \lambda).o} \tag{3.46}$$

Here the sum is extended over all production rules $A \rightarrow \lambda$ in the grammar. This is used in the convergence criterion:

$$|(HH(iteration) - HH(iteration - 1)| < \varepsilon \tag{3.47}$$

Suppose the database is composed by only the sequence of Fig. 8. Then the values of the raw and normalized observed histograms (N and N') will be:

$$\begin{aligned}
&N(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1\\
&N(Start \rightarrow \texttt{Na}\ \epsilon).o = 1\\
&N(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1\\
&N(\texttt{Acq} \rightarrow Q6\ Q6).o = 1\\
&N(\texttt{Na} \rightarrow Q6\ Q6).o = 1\\
&N(\texttt{Tm} \rightarrow Q6\ Q6).o = 1\\
&N(Q6 \rightarrow W6\ W6).o = 2\\
&N(W6 \rightarrow 6).o = 4\\
&N(\epsilon \rightarrow 10).o = 4
\end{aligned} \tag{3.48}$$

and

$$\begin{aligned}
&N'(Start \rightarrow \texttt{Acq}\ \epsilon).o = 1/3\\
&N'(Start \rightarrow \texttt{Na}\ \epsilon).o = 1/3\\
&N'(Start \rightarrow \texttt{Tm}\ \epsilon).o = 1/3\\
&N'(\texttt{Acq} \rightarrow Q6\ Q6).o = 1\\
&N'(\texttt{Na} \rightarrow Q6\ Q6).o = 1\\
&N'(\texttt{Tm} \rightarrow Q6\ Q6).o = 1
\end{aligned}$$

$$N'(Q6 \rightarrow W6\ W6).o = 1$$
$$N'(W6 \rightarrow 6).o = 1$$
$$N'(\epsilon \rightarrow 10).o = 1 \tag{3.49}$$

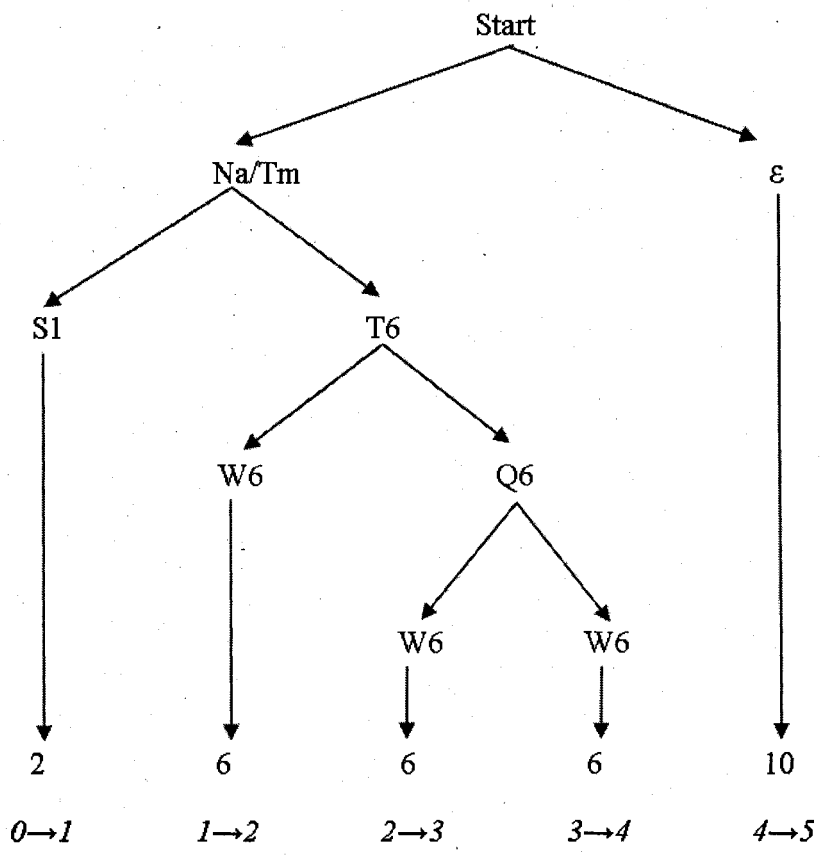

Figure 13 Example of derivation tree and sequence generated by the grammar for Mercury.

All the other rule frequencies are set to 0. Suppose the grammar whose probabilities are presented below generates the sequence of Fig. 13. Note that some production probabilities are not involved in this example, and that for consistency, $\sum_\lambda \theta(A \rightarrow \lambda) = 1$.

$$\theta(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1$$
$$\theta(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1$$
$$\theta(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1$$
$$\theta(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667$$
$$\theta(\texttt{Na} \rightarrow Q6\ Q6) = 0.5$$
$$\theta(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408$$
$$\theta(\texttt{Na} \rightarrow S1\ T6) = 0.5$$
$$\theta(\texttt{Tm} \rightarrow S1\ T6) = 0.1408$$
$$\theta(T6 \rightarrow W6\ Q6) = 1$$
$$\theta(Q6 \rightarrow W6\ W6) = 1$$
$$\theta(S1 \rightarrow 2) = 0.2$$
$$\theta(W6 \rightarrow 6) = 1$$
$$\theta(\epsilon \rightarrow 10) = 1 \qquad (3.50)$$

The values of the raw and normalized sample histograms (N and N') for this sequence will be:

$$N(Start \rightarrow \texttt{Acq}\ \epsilon).s = 0$$
$$N(Start \rightarrow \texttt{Na}\ \epsilon).s = 1$$
$$N(Start \rightarrow \texttt{Tm}\ \epsilon).s = 1$$
$$N(\texttt{Acq} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Na} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Tm} \rightarrow Q6\ Q6).s = 0$$
$$N(\texttt{Na} \rightarrow S1\ T6).s = 1$$

$$
\begin{aligned}
&N(\texttt{Tm} \rightarrow S1\ T6).s = 1 \\
&N(T6 \rightarrow W6\ Q6).s = 1 \\
&N(Q6 \rightarrow W6\ W6).s = 1 \\
&N(S1 \rightarrow 2).s = 1 \\
&N(W6 \rightarrow 6).s = 3 \\
&N(\epsilon \rightarrow 10).s = 1
\end{aligned} \tag{3.51}
$$

and

$$
\begin{aligned}
&N'(Start \rightarrow \texttt{Acq}\ \epsilon).s = 0 \\
&N'(Start \rightarrow \texttt{Na}\ \epsilon).s = 1/2 \\
&N'(Start \rightarrow \texttt{Tm}\ \epsilon).s = 1/2 \\
&N'(\texttt{Acq} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Na} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Tm} \rightarrow Q6\ Q6).s = 0 \\
&N'(\texttt{Na} \rightarrow S1\ T6).s = 1 \\
&N'(\texttt{Tm} \rightarrow S1\ T6).s = 1 \\
&N'(T6 \rightarrow W6\ Q6).s = 1 \\
&N'(Q6 \rightarrow W6\ W6).s = 1 \\
&N'(S1 \rightarrow 2).s = 1 \\
&N'(W6 \rightarrow 6).s = 1 \\
&N'(\epsilon \rightarrow 10).s = 1
\end{aligned} \tag{3.52}
$$

Thus, setting χ to 0.1, the probabilities are re-estimated using Eq. 3.44 as follows:

$$\theta'(Start \rightarrow \texttt{Acq}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 0)) = 0.103$$

$$\theta'(Start \rightarrow \texttt{Na}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 1/2)) = 0.098$$
$$\theta'(Start \rightarrow \texttt{Tm}\ \epsilon) = 0.1 \cdot (1 + 0.1 \cdot (1/3 - 1/2)) = 0.098$$
$$\theta'(\texttt{Acq} \rightarrow Q6\ Q6) = 0.1667 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.18337$$
$$\theta'(\texttt{Na} \rightarrow Q6\ Q6) = 0.5 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.55$$
$$\theta'(\texttt{Tm} \rightarrow Q6\ Q6) = 0.1408 \cdot (1 + 0.1 \cdot (1 - 0)) = 0.15488$$
$$\theta'(\texttt{Na} \rightarrow S1\ T6) = 0.5 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.45$$
$$\theta'(\texttt{Tm} \rightarrow S1\ T6) = 0.1408 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.12672$$
$$\theta'(T6 \rightarrow W6\ Q6) = 1 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.9$$
$$\theta'(Q6 \rightarrow W6\ W6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1$$
$$\theta'(S1 \rightarrow 2) = 0.2 \cdot (1 + 0.1 \cdot (0 - 1)) = 0.18$$
$$\theta'(W6 \rightarrow 6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1$$
$$\theta'(\epsilon \rightarrow 6) = 1 \cdot (1 + 0.1 \cdot (1 - 1)) = 1 \tag{3.53}$$

The probabilities should thereafter be normalized over all the production probabilities, including those which are not involved in the example, to be consistent.

HOLA's time complexity per iteration only depends on the size of the histograms and is constant with regard to the amount of data. HOLA also has a very low memory complexity. This algorithm has a time complexity and a memory complexity that are both of $O(M_{nt}^3)$. On the other hand, the number of iterations will vary with regard to parameters, such as the number of sequences randomly generated, the value of χ and the initial distribution of the production probabilities.

3.4 Comparison of learning techniques

It has already been mentioned that gEM and TS were numerically equivalent. Moreover, gEM(IO) and TS(IO) are numerically equivalent to the classical IO algorithm, while

gEM(VS) and TS(VS) are numerically equivalent to the classical VS algorithm. The differences between these two techniques lie in both pre-processing and iterative procedures.

Pre-processing with TS results in a non-compact representation of the derivation trees of the sequences of the training dataset. This allows for very fast re-estimation of the data during the iterative process. However, the derivation trees can be listed only if the grammar is low ambiguous, which limits the total number of derivation trees for a given sequence, otherwise memory requirements become an issue. Pre-processing with gEM creates support graphs, which is a more compact representation of the derivation trees, and thereby allows dealing with more ambiguous grammar, but resulting in a higher time complexity per iteration for low-ambiguity grammars.

HOLA is very different from the other algorithms, although it also uses a parser for pre-processing of the training dataset. Pre-processing in this case results in an histogram summarizing the frequency of appearance of the rules during the parsing, and therefore has a size that only depends from the number of non-terminals of the grammar, which is very low for radar ES applications. Its time complexity also depends only from the number of non-terminals of the grammar, and the iterative process is therfore much faster than those of TS and gEM. However, it has the main inconvenience of not optimizing the likelihood of the training dataset, which may lead to lower accuracy than TS and gEM.

Table I displays a summary of the re-estimation formulas of the above-described techniques, namely IO, VS, TS(IO), TS(VS), gEM(IO), gEM(VS) and HOLA, along with their relation with the global re-estimation formula of Eq.3.2. This table contains definitions of the symbols used in each reestimation formula. IO re-estimates the production rule probabilities using inside and outside probabilities, while VS does this using a simplification of Eq. 3.2. TS(IO) uses Eq. 3.2 directly, while TS (VS) uses the same equation as VS. gEM(IO) and gEM(VS) re-estimate the probabilities by normalizing the η and $\hat{\eta}$. Note,

however, that TS/gEM(IO) and TS/gEM(VS) all have equivalent formulas. In contrast, HOLA uses an approximation of the gradient descent.

Table I

Re-estimation formulas of techniques for batch learning of SCFGs.

Method	Reestimation Formula
General	$\theta'(A \to \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x \mid G_s)}{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}$ $N(A \to \lambda, d_x)$ = frequency of the rule $A \to \lambda$ in the derivation tree d_x
IO	$\theta'(A \to BC) = \frac{\sum_{x \in \Omega} \frac{\sum_{0 \le i < k < j \le L} \alpha(i,k \mid B)\alpha(k,j \mid C)\beta(i,j \mid A)\theta(A \to BC)}{\alpha(0,L \mid Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j \mid A)\beta(i,j \mid A)}{\alpha(0,L \mid Start)}}$ $\theta'(A \to a) = \frac{\sum_{x \in \Omega} \frac{\sum_{i \mid w_i = a} \beta(i-1,i \mid A)\theta(A \to a)}{\alpha(0,L \mid Start)}}{\sum_{x \in \Omega} \frac{\sum_{0 \le i < j \le L} \alpha(i,j \mid A)\beta(i,j \mid A)}{\alpha(0,L \mid Start)}}$ $\alpha(i,j \mid A)$=inside probability of A generating the subsequence $w_{i+1}, ..., w_j$ $\beta(i,j \mid A)$=outside probability of A generating the subsequence $w_{i+1}, ..., w_j$
VS	$\theta'(A \to \lambda) = \frac{N(A \to \lambda, \hat{d}_x)}{N(A, \hat{d}_x)}$ $\hat{d}_x$ = best derivation tree of sequence x
TS(IO)	$\theta'(A \to \lambda) = \frac{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A \to \lambda, d_x) P(x, d_x \mid G_s)}{\sum_{x \in \Omega} \frac{1}{P(x,\Delta_x \mid G_s)} \sum_{d_x \in \Delta_x} N(A, d_x) P(x, d_x \mid G_s)}$

Method	Reestimation Formula
TS(VS)	$\theta'(A \rightarrow \lambda) = \frac{N(A \rightarrow \lambda, \bar{d}_x)}{N(A, \bar{d}_x)}$
gEM(IO)	$\theta'(A \rightarrow \lambda) = \frac{\eta(A \rightarrow \lambda)}{\sum_\lambda \eta(A \rightarrow \lambda)}$ $\eta(A \rightarrow \lambda)$=sum on all the strings of the normalized balanced frequency of the rule $A \rightarrow \lambda$
gEM(VS)	$\theta'(A \rightarrow \lambda) = \frac{\hat{\eta}(A \rightarrow \lambda)}{\sum_\lambda \hat{\eta}(A \rightarrow \lambda)}$ $\hat{\eta}(A \rightarrow \lambda)$=$N(A \rightarrow \lambda, \hat{d}_x)$
HOLA	$\theta'(A \rightarrow \lambda) = \theta(A \rightarrow \lambda) \cdot (1 + \chi \cdot (N(A \rightarrow \lambda).o - N(A \rightarrow \lambda).s))$ $N(A \rightarrow \lambda).o$ = frequency of the rule $A \rightarrow \lambda$ over the training set $N(A \rightarrow \lambda).s$ = frequency of the rule $A \rightarrow \lambda$ over the generated set