Stochastic inference of regular tree languages *

Submitted to the Special Issue of the Machine Learning Journal on Automata Induction, Grammar Inference, and Language Acquisition

Rafael C. Carrasco (carrasco@dlsi.ua.es), Jose Oncina (oncina@dlsi.ua.es) and Jorge Calera-Rubio (calera@dlsi.ua.es)
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante

1. Introduction

A common concern in grammatical inference tasks is to avoid overgeneralization. Although complete samples (that is, samples that include information about examples and counter-examples) may be used for this purpose, representative sets of counter-examples are usually difficult to obtain. A different way to prevent overgeneralization is the use of stochastic samples. Indeed, many experimental settings involve random or noisy examples. Angluin (1988) proved that identification in the limit from stochastic samples is possible if rather general assumption about the probability distribution are made. Some algorithms for learning string regular languages from stochastic samples have been proposed before (Stolcke and Omohundro, 1993; Carrasco and Oncina, 1998). The last one has the interesting property that identification in the limit of the structure of the deterministic automaton (DFA) is guaranteed.

Learning context-free languages is harder, but identification of the grammar is still possible if structural descriptions are available. In such case, the identification of CFG's becomes equivalent to the problem of identifying regular tree languages, and algorithms for this purpose have been proposed by Sakakibara (1992) and Oncina and García (1994). The first algorithm uses positive examples and works within the subclass of reversible tree languages. The second one uses complete samples but identifies any deterministic tree grammar (or equivalently, any backwards deterministic CFG).

In this paper, we introduce a modification of the last algorithm that can be trained with positive samples generated according to a probabilistic production scheme. The construction follows the same guidelines as the algorithm for string languages in Carrasco and Oncina (1998). A different approach (Sakakibara et al., 1994) generalizes the

* Work partially supported by the Spanish CICYT under grant TIC97-0941.

forward-backward algorithm used to train hidden Markov models and uses some aditional information as the number of parameters in the model or a rough grammar to initialize the algorithm.

The basic notation is described in sections 2 and 3 and the algorithm is described in sections 4 and 5. In section 6, we briefly describe a method to directly evaluate the relative entropy between the inferred language and the true grammar which avoids the generation of huge test sets. Finally an example is presented in section 7.

2. Regular tree languages

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}^*$ be the free monoid generated by $\mathbb{N}$ with "." as the operation and λ as the identity. A subset $D \subseteq \mathbb{N}^*$ is a *tree domain* if for all $x, y \in \mathbb{N}^*$ and for all $i, j \in \mathbb{N}$

$$\begin{aligned} x.y \in D &\Rightarrow x \in D \\ x.i \in D \wedge j \leq i &\Rightarrow x.j \in D \end{aligned} \tag{1}$$

A *finite tree* over the alphabet V is a mapping $t : \mathrm{dom}(t) \to V$ where $\mathrm{dom}(t)$ is a finite tree domain. The *depth* of a tree is the maximum length of the elements in its domain.

Given an alphabet V, the set of all finite trees whose nodes are labeled with symbols in V will be denoted with V^T. In the following, finite trees will be represented using the functional notation: for instance, the functional notation for the tree shown in Fig. 1 is $a(b(a(bc))c)$. In particular, any symbol a in V is also the representation of a particular tree t such that $\mathrm{dom}(t) = \{\lambda\}$ and $t(\lambda) = a$ and, thus, one can write $V \subset V^T$.

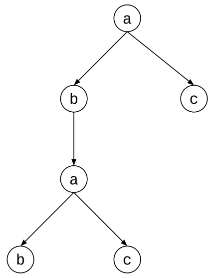

Figure 1. A graphic representation of the tree $a(b(a(bc))c)$

Deterministic tree automata (DTA) generalize deterministic finite-state automata (DFA) that work on strings. In contrast with DFA —where strings are processed from left to right—, in DTA the trees are processed bottom-up and a state in the automaton is assigned to every node in the tree. This state depends on the node label and on the states associated to the siblings of the node. The state assigned to the root of the tree has to be an accepting state for the tree to be accepted by the automaton. Formally, a DTA is a 4-tuple $A = (Q, V, \delta, F)$, where

- Q is a finite set of *states*;
- V is a finite set of *labels*;
- $F \subset Q$ is the subset of *accepting states*;
- $\delta = \{\delta_0, \delta_1, ..., \delta_n\}$ is a set of *transition functions* of the form $\delta_k : V \times Q^k \to Q$.

If $t = f(t_1, t_2, ..., t_k)$ is a tree (or subtree) consisting of an internal node labeled f which expands k subtrees $t_1, t_2, ..., t_k$, the state $\delta(t)$ is $\delta_k(f, \delta(t_1), \ldots, \delta(t_k))$. In other words, $\delta(t)$ is recursively defined as:

$$\delta(t) = \begin{cases} \delta_k(f, \delta(t_1), \ldots, \delta(t_k)) & \text{if} \quad t = f(t_1 \ldots t_k) \in (V^T - V) \\ \delta_0(a) & \text{if} \quad t = a \in V \end{cases} \tag{2}$$

Every DTA defines a *regular tree language* (RTL) consisting of all trees accepted by the automaton: $L(A) = \{t \in V^T : \delta(t) \in F\}$. By convention, undefined transitions lead to *absorption* states, i.e., to non-acceptable trees. DTA are closely connected to context-free grammars. For instance, given a backwards-deterministic (Aho and Ullman, 1972) grammar $G = (N, \Sigma, P, S)$, the acceptor

$$\begin{aligned} Q &= N \cup \Sigma \\ V &= \{\lambda\} \cup \Sigma \\ \delta_0(a) &= a \qquad\qquad \forall a \in \Sigma \\ \delta_k(\lambda, X_1, ..., X_k) &= A \qquad\qquad \text{if } A \to X_1...X_k \in P \end{aligned} \tag{3}$$

recognizes the set of skeletal derivation trees of G.

3. Stochastic tree automata

Stochastic tree automata incorporate a probability for every transition in the automaton, with the normalization that the probabilities of transitions leading to the same state $q \in Q$ must add up to one. In other words, there is a collection of functions $p = \{p_0, p_1, p_2, ..., p_n\}$ of the type $p_k : V \times Q^k \to [0, 1]$ such that they satisfy, for all $q \in Q$,

$$\sum_{f \in V} \sum_{k=0}^{n} \sum_{\substack{q_1, ..., q_k \in Q: \\ \delta_k(f, q_1, ..., q_k) = q}} p_k(f, q_1, ..., q_k) = 1 \tag{4}$$

In addition to this probabilities, every *stochastic deterministic tree automaton* $A = (Q, V, \delta, p, r)$ provides a function $r : Q \to [0, 1]$ which, for every $q \in Q$, gives the probability that a tree satisfies $\delta(t) = q$ and replaces, in the definition of the automaton, the subset of accepting states. Then, the probability of a tree t in the language generated by A is given by the product of the probabilities of all the transitions used when t is processed by A, times $r(\delta(t))$:

$$p(t|A) = r(\delta(t))\ \pi(t) \tag{5}$$

with $\pi(t)$ recursively given by

$$\pi(f(t_1, \ldots, t_k)) = p_k(f, \delta(t_1), \ldots, \delta(t_k))\ \pi(t_1) \cdots \pi(t_k)\ . \tag{6}$$

Of course, $\pi(a) = p_0(a)$ if $t = a \in V$. The equations (5) and (6) define a probability distribution $p(t|A)$ which is consistent if

$$\sum_{t \in V^T} p(t|A) = 1. \tag{7}$$

The condition of consistency can be written in terms of the expectation elements:

$$\Lambda_{ij} = \sum_{f \in V} \sum_{k=1}^{n} \sum_{\substack{i_1, i_2, \ldots, i_k \in Q: \\ \delta(f, i_1, \ldots, i_k) = j}} p_k(f, i_1, i_2, ..., i_k)(\delta_{ii_1} + \delta_{ii_2} + \cdots + \delta_{ii_k})\ , \tag{8}$$

where δ_{ij} is Kronecker's delta. Consistency is preserved if the spectral radius of matrix Λ is smaller than one (Wetherell, 1980).

It is important to remark that two stochastic languages are *identical* if

$$T_1 = T_2 \Leftrightarrow p(t|T_1) = p(t|T_2)\ \ \forall t \in V^T. \tag{9}$$

In contrast to strings, concatenation of trees requires marking the node where the attachment takes place. For this purpose, let \$ be a special symbol not in V. With $V_\$^T$ we denote the set of trees in $(V \cup \{\$\})^T$ with no internal node labeled \$ and exactly one leaf labeled with \$. For every $s \in V_\$^T$ and every $t \in V^T \cup V_\$^T$, the tree st is obtained replacing in s the \$-marked node with a copy of t.

For every stochastic tree language T and for every $t \in V^T$, the *quotient* T/t is the stochastic language over $V_\$^T$ defined through the probabilities

$$p(s|T/t) = \frac{p(st|T)}{p(V_\$^T t|T)}\ . \tag{10}$$

If $p(V_\$^T t|T) = 0$, the quotient (10) is undefined and in such case, we will write by convention $T/t = \emptyset$.

The Myhill-Nerode's theorem for regular languages (Hopcroft and Ullman, 1980) can be generalized for regular tree languages (Gécseg and Steinby, 1984) as well as for stochastic languages (Carrasco and Oncina, 1998). If T is a stochastic RTL, the number of different sets T/t is finite and a DTA accepting $\{t \in V^T : p(t|T) > 0\}$ can be defined. We will call it the *canonical acceptor* $M = (Q^M, V, \delta^M, F^M)$:

$$\begin{aligned} Q^M &= \{T/t \neq \emptyset : t \in V^T\} \\ F^M &= \{T/t : p(t|T) > 0\} \\ \delta^M(f, T/t_1, \ldots, T/t_k) &= T/f(t_1, \ldots, t_k) \end{aligned} \tag{11}$$

A stochastic sample S of the language T is an infinite sequence of trees generated according to the probability distribution $p(t|T)$. We denote with S_n the sequence of the n first trees (not necessarily different) in S and with $c_n(t)$ the number of occurrences of tree t in S_n. For $X \subset V^T$, $c_n(X) = \sum_{t \in X} c_n(t)$. If the structure (that is, the states and transition functions) of M is known, we can transform M into a stochastic DTA by estimating the probability functions from the examples in S_n, that is,

$$\begin{aligned} r^M(T/t) &\simeq \frac{c_n([t])}{n} \\ p_k^M(f, T/t_1, \ldots, T/t_k) &\simeq \frac{c_n(V_\$^T f([t_1], \ldots, [t_k]))}{c_n(V_\$^T [f(t_1, \ldots, t_k)])} \ . \end{aligned} \tag{12}$$

where $[t]$ is the equivalence class defined by the relation $s \equiv t \Leftrightarrow \delta^M(s) = \delta^M(t)$ and $f([t_1], ..., [t_k])) = \{f(s_1, ..., s_k) : s_i \in [t_i]\}$

4. Inference algorithm

In the following, we will assume that an arbitrary total order relation has been defined in V^T such that $t_1 \leq t_2 \Leftrightarrow \mathrm{depth}(t_1) \leq \mathrm{depth}(t_2)$.

In order to identify M, the *subtree set* and the *short-subtree set* are respectively defined as

$$\begin{aligned} \mathrm{Sub}(T) &= \{t \in V^T : T/t \neq \emptyset\} \\ \mathrm{SSub}(T) &= \{t \in \mathrm{Sub}(T) : \forall s \in \mathrm{Sub}(T)\ T/s = T/t \Rightarrow t \leq s\} \end{aligned} \tag{13}$$

Note that there is exactly one tree in $\mathrm{SSub}(T)$ for every state $q \in Q^M$ of the canonical acceptor M and, therefore, the subtrees in $\mathrm{SSub}(T)$ can be regarded as representatives of the states in Q^M. If $t_1, ..t_k$ and $f(t_1, ..., t_k)$ are in $\mathrm{SSub}(T)$ then there is a transition in M represented

```
algorithm for Tree Language Inference
input: A ⊂ Sub(T) such that K(T) ⊂ A
output: SSub (short subtree set)
        F (frontier set)
begin algorithm
  SSub = F = ∅
  W = V ∩ A                              // candidates temporary storage
  do ( while W ≠ ∅ )
    x = min W                            // x = f(t_1, ..., t_k)
    W = W − {x}
    if ∃y ∈ SSub : equiv_T(x, y) then
      F = F ∪ {x}                        // x is not a short subtree
      δ^M(f, t_1, ..., t_k) = y
    else
      SSub = SSub ∪ {x}                  // x is a short subtree
      W = W ∪ {f(t_1, ..., t_k) ∈ A : t_1, ..., t_k ∈ SSub}   // update W
      δ^M(f, t_1, ..., t_k) = x
    endif
  end do
end algorithm
```

Figure 2. Inference algorithm.

by $\delta^M(f, t_1, ..., t_k) = f(t_1, ..., t_k)$. For those cases where $f(t_1, ..., t_k)$ is not in $\mathrm{SSub}(T)$, we define the *kernel* and the *frontier set* as follows:

$$\begin{aligned} K(T) &= \{f(t_1, ..., t_k) \in \mathrm{Sub}(T) : t_1, ..., t_k \in \mathrm{SSub}(T)\} \\ F(T) &= K(T) - \mathrm{SSub}(T) \end{aligned} \tag{14}$$

Every tree in $K(T) - V$ is tied to a transition in M and, therefore, $K(T)$ is finite.

Finally, we define a boolean function $\mathtt{equiv}_{\mathtt{T}} : K(T) \times K(T) \to \{\text{TRUE}, \text{FALSE}\}$ such that

$$\mathtt{equiv}_{\mathtt{T}}(t_1, t_2) = \text{TRUE} \Leftrightarrow T/t_1 = T/t_2. \tag{15}$$

The following theorems support the inference algorithm:

THEOREM 1. *If* $\mathrm{SSub}(T)$, $K(T)$ *and* $\mathtt{equiv}_{\mathtt{T}}$ *are known, then the structure of the canonical acceptor is isomorphic to:*

$$\begin{aligned} Q &= \mathrm{SSub}(T) \\ \delta(f, t_1, ..., t_k) &= t \quad \forall f(t_1, ..., t_k) \in K(T) \end{aligned} \tag{16}$$

where t *is the only tree in* $\mathrm{SSub}(T)$ *satisfying* $\mathtt{equiv}_{\mathtt{T}}(t, f(t_1, ..., t_k))$.

Proof. Let $\Phi : Q \to Q^M$ be the mapping such that, for every tree t in $Q = \mathrm{SSub}(T)$, $\Phi(t) = T/t$. The mapping Φ is an isomorphism if

$$\Phi(\delta(f, t_1, ..., t_k)) = \delta^M(f, \Phi(t_1), ..., \Phi(t_k)),$$

or, looking at (11),

$$T/\delta(f, t_1, ..., t_k) = T/f(t_1, ..., t_k).$$

That is, Φ is an isomorphism if and only if $\delta(f, t_1, ..., t_k)$ is a subtree $t \in \mathrm{SSub}(T)$ such that $\mathtt{equiv_T}(t, f(t_1, ..., t_k))$ and, according to the definition (13), t is unique.

THEOREM 2. *The algorithm in Fig. 2 outputs* $\mathrm{SSub}(T)$*,* $F(T)$ *and* δ^M *with input* $\mathtt{equiv_T}$ *plus any* $A \subset \mathrm{Sub}(T)$ *such that* $K(T) \subset A$*.*

Proof. Simple induction in i shows that after i iterations $\mathrm{SSub}^{[i]} \subset \mathrm{SSub}(T)$, $F^{[i]} \subset F(T)$ and $W^{[i]} \subset K(T)$. On the other hand, if $t \in K(T)$ then $t \in A$ and induction in the depth of the tree shows that t eventually enters the algorithm. From theorem 1, for every $t_1, ..., t_k \in \mathrm{SSub}(T)$, if $f(t_1, ..., t_k) \in \mathrm{SSub}(T)$ then $\delta(f, t_1, ..., t_k) = f(t_1, ..., t_k)$. However, if $f(t_1, ..., t_k) \notin \mathrm{SSub}(T)$, there exists a single tree $s \in \mathrm{SSub}(T)$ satisfying $\mathtt{equiv_T}(s, f(t_1, ..., t_k))$ and then $\delta(f, t_1, ..., t_k) = s$.

Note that the finite set $\mathrm{Sub}(S_n) \subset \mathrm{Sub}(T)$ can be used as input in the former algorithm, as $K(T) \subset \mathrm{Sub}(S_n)$ for n large enough. On the other hand, the algorithm never calls $\mathtt{equiv_T}$ out of its domain $K(T)$ and thus, the number of calls is bounded by $|K(T)|^2$. Thus, the global complexity of the algorithm is $\mathcal{O}(|K(T)|^2)$ times the complexity of function $\mathtt{equiv_T}$.

5. Probabilistic inference

According to the definition (15), two trees x and y are equivalent if $T/x = T/y$. This can be checked by means of Eq. (10) or, equivalently,

$$p(V_\$^T t | T/x) = p(V_\$^T t | T/y) \; \forall t \in V^T \tag{17}$$

However, in order to improve convergence, we rather check the conditional probabilities:

$$p(V_\$^T tz | T/(zx)) = p(V_\$^T tz | T/(zy)) \; \forall t \in V_{\$1}^T \; \forall z \in V_\$^T \tag{18}$$

where $V_{\$1}^T$ is the subset made of those trees in $V_\$^T$ whose \$-marked node is at depth one. In practice, the target language T is replaced by

```
algorithm comp_n
input: x, y ∈ V^T, S_n
output: boolean
begin algorithm
  do ( ∀z ∈ V_$^T : (zx ∨ zy) ∈ Sub(S_n), ∀t ∈ V_$1^T )
     if differ(c_n(V_$^T tzx), c_n(V_$^T zx), c_n(V_$^T tzy), c_n(V_$^T zy), α) then
        return FALSE
     endif
  end do
  return TRUE
end algorithm
```

Figure 3. Algorithm $\mathtt{comp_n}$.

the stochastic sample S and the equivalence test is performed through a probabilistic function $\mathtt{comp_n}(x, y)$ (Fig. 3) of the n first trees in S, i.e., of S_n. The algorithm will output the correct DTA in the limit as long as $\mathtt{comp_n}$ converges to $\mathtt{equiv_T}$ when n grows. For the statistical checks, we have chosen a Hoeffding (1963) type test, as described in Fig. 4. This test provides the correct answer with probability greater than $(1-\alpha)^2$, α being an arbitrarily small positive number. Because the number of checks performed by the algorithm grows with the number of different subtrees in the sample, we allow the parameter α to depend on n. Next theorem shows that α_n can be selected in such a way that $\mathtt{comp_n}$ converges to $\mathtt{equiv_T}$.

THEOREM 3. *Let α_n be the parameter used in function* `differ` *(Fig. 4) and $\sum_{n=1}^{\infty} n\alpha_n$ be finite. Then, with probability one,* $\mathtt{comp_n}(x, y) = \mathtt{equiv_T}(x, y)$ *for all $x, y \in K(T)$ except for finitely many values of n.*

Proof. As `differ` works with confidence level $(1-\alpha)^2$, the algorithm $\mathtt{comp_n}$ plotted in Fig. 3 returns the correct value with probability greater than $(1-\alpha_n)^{2\tau_n}$, where τ_n is the number of different subtrees in S_n. Then, the probability p_n that $\mathtt{equiv_T}(x, y) \neq \mathtt{comp_n}(x, y)$ is smaller than $2\alpha_n\tau_n$. If $\sum_n p_n$ is finite then, the Borel-Cantelli lemma (Feller, 1950) guarantees that, with probability one, only a finite number of mistakes take place. The expected number of different subtrees τ_n grows at most linearly with n and then, $\sum_{n=1}^{\infty} n\alpha_n < \infty$ is a sufficient condition.

Note that the complexity of $\mathtt{comp_n}$ is at most $\mathcal{O}(n)$ while $|K(T)|$ does not depend on S_n and therefore, the global complexity of the algorithm remains $\mathcal{O}(n)$.

```
algorithm differ
input: f, m, f', m', α
output: boolean
begin algorithm
  return |f/m − f'/m'| > sqrt(1/(2m) log 2/α) + sqrt(1/(2m') log 2/α)
end algorithm
```

$$\textbf{return}\ \left|\frac{f}{m}-\frac{f'}{m'}\right| > \sqrt{\frac{1}{2m}\log\frac{2}{\alpha}} + \sqrt{\frac{1}{2m'}\log\frac{2}{\alpha}}$$

Figure 4. Algorithm `differ`.

6. Relative entropy between stochastic tree languages

In this section we briefly describe our procedure to compute the relative entropy between two stochastic DTA. A more detailed description can be found in Calera-Rubio and Carrasco (1998). Given a probability distribution $p(t|A')$ over V^T that approximates the true one $p(t|A)$, the magnitude

$$G(A, A') = -\sum_{t \in V^T} p(t|A) \log_2 p(t|A') \tag{19}$$

bounds, within a deviation of one bit (Cover and Thomas, 1991), the average length of the string needed to code a tree in V^T provided that the information contained in A' and an optimal coding scheme are used. The difference $H(A, A') = G(A, A') - G(A, A)$ gives the *relative entropy* or *Kullback-Leibler distance* between A and A'.

Recall from Eq. (5) that the probability that the tree t is generated by the automaton $A' = (Q', V, \delta', p', r')$ is given by the product of two different factors, and then $\log_2 p(t|A') = \log_2 r'(\delta'(t)) + \log_2 \pi'(t)$. The contribution to $G(A, A')$ of the r-terms is (Calera-Rubio and Carrasco, 1998)

$$G_r(A, A') = -\sum_{\substack{i \in Q \\ j \in Q'}} r(i) \eta_{ij} \log_2 r'(j) \tag{20}$$

where η_{ij} represents the probability that a node of type $i \in Q$ expands as a subtree t such that $\delta'(t) = j$ and can be easily obtained by means of an iterative procedure:

$$\eta_{ij}^{[t+1]} = \sum_{f \in V} \sum_{k=0}^{n} \sum_{\substack{i_1, i_2, \ldots, i_k \in Q: \\ \delta_k(f, i_1, i_2, \ldots, i_k) = i}} \sum_{\substack{j_1, j_2, \ldots, j_k \in Q': \\ \delta'_k(f, j_1, j_2, \ldots, j_k) = j}} p_k(f, i_1, i_2, \ldots, i_k)\, \eta_{i_1 j_1}^{[t]} \eta_{i_2 j_2}^{[t]} \cdots \eta_{i_k j_k}^{[t]} \tag{21}$$

with $\eta_{ij}^{[0]} = 0$.

On the other hand, the contribution to $G(A, A')$ of the π-terms can be written as

$$G_\pi(A, A') = -\sum_{f \in V} \sum_{k=0}^{n} \sum_{i_1, i_2, \ldots, i_k \in Q} \sum_{j_1, j_2, \ldots, j_k \in Q'} C_{\delta_k(f, i_1, i_2, \ldots, i_k)} \times$$
$$p_k(f, i_1, i_2, \ldots, i_k) \log_2 p'_k(f, j_1, j_2, \ldots, j_k) \eta_{i_1 j_1} \eta_{i_2 j_2} \cdots \eta_{i_k j_k} \tag{22}$$

where C_i is the expected number of subtrees of type i in a tree generated according to $p(t|A)$. This vector $\mathbf{C}$ of expectation values C_i can be easily computed using the matrix Λ defined in Eq. (8) together with the vector $\mathbf{r}$ of probabilities $r(i)$. As shown in Wetherell (1980), $\mathbf{C} = (\sum_{m=0}^{\infty} \Lambda^m)\, \mathbf{r}$ and, then, $\mathbf{C} = \mathbf{r} + \Lambda \mathbf{C}$. This relationship allows a fast iterative computation:

$$C_i^{[t+1]} = r(i) + \sum_{j \in Q} \Lambda_{ij} C_j^{[t]} \tag{23}$$

with $C_i^{[0]} = 0$. As in the case of Eq. (21), it is straightforward to show that the iterative procedure converges monotonically to the correct value.

7. An example

To illustrate the application of the algorithm we show some results for a simple tree grammar, which generates parse trees of conditional statements:

$$\begin{array}{rcll}
\delta_7\ (\textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ S\ \textbf{endif}) & = & S & (0.2) \\
\delta_5\ (\textbf{if}\ E\ \textbf{then}\ S\ \textbf{endif}) & = & S & (0.2) \\
\delta_2(\textbf{print}\ E) & = & S & (0.6) \\
\delta_3(E\ \textbf{operator}\ T) & = & E & (0.3) \\
\delta_1(T) & = & E & (0.7) \\
\delta_3(T\ \textbf{exp}\ \textbf{n}) & = & T & (0.1) \\
\delta_1(F) & = & T & (0.9) \\
\delta_3(\textbf{(}E\textbf{)}) & = & F & (0.8) \\
\delta_1(\textbf{n}) & = & F & (0.2)
\end{array}$$

Variables appear in uppercase, terminals in bold and the number in parenthesis represents the probability of the transition. The average

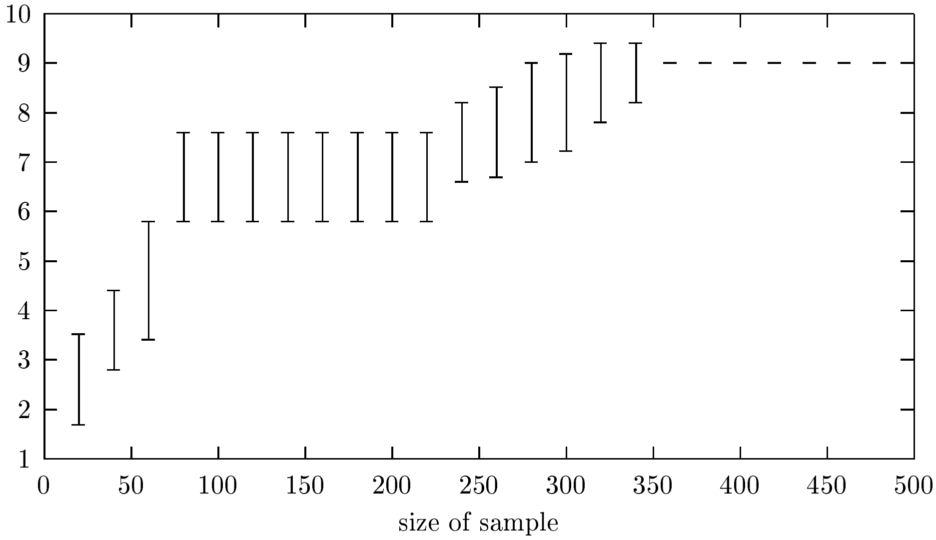

Figure 5. Average number of transition rules in the hypothesis as a function of the number of examples. The target grammar has 9 rules.

number of transition rules in the hypothesis as a function of the number of examples is plotted in Fig. 5. When the sample is small, rather small grammars are found and overgeneralization occurs. As the number of examples grows, the algorithm tends to output a grammar with the correct size, and for larger samples (above 500 examples) the correct grammar is always found. Our implementation needed very few seconds to process the sample, even when it contained thousands of examples. In Fig. 6, the relative entropy between the target grammar and the hypothesis is computed following the method described in former section. The results are shown in the region where identification takes place and the relative entropy becomes always finite. For comparison purposes, the relative entropy between the target grammar and the sample is also plotted. It is clear from the figure that identifying the structure of the DTA makes the distance converge much faster than a mere estimation of the probabilities from the sample.

8. Conclusions

We have developed an algorithm that learns deterministic regular tree grammars from stochastic examples in linear time with the size of the sample. It identifies the minimal stochastic automaton generating the language and no counter-examples are used. Experimentally, identi-

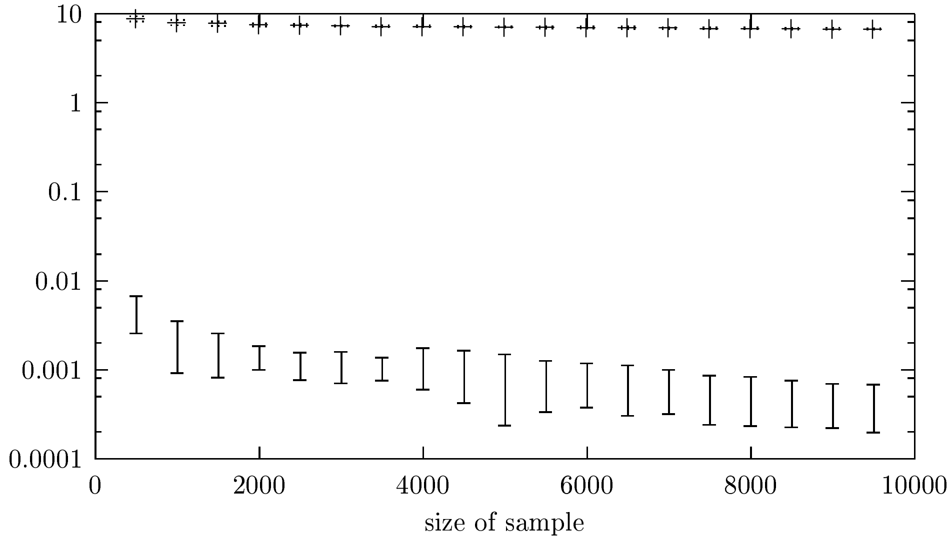

Figure 6. Lower dots: relative entropy between the target grammar and the output of the algorithm as a function of the number of examples in the sample. Upper dots: relative entropy between the target grammar and the sample.

fication is reached with relatively small samples, the relative entropy between the model and the target grammar decreases very fast with the size of the sample, and the algorithm proves fast enough for application purposes.

References

Aho, A. V. and J. D. Ullman: 1972, *The Theorie of Parsing, Translation and Compiling - Vol.I: Parsing.* London: Prentice Hall.

Angluin, D.: 1988, 'Identifying languages from stochastic examples'. Technical Report YALEU/DCS/RR-614, Yale University Dept. of Computer Science, New Haven, CT.

Calera-Rubio, J. and R. C. Carrasco: 1998, 'Computing the relative entropy between regular tree languages'. *Information Processing Letters* **to appear**.

Carrasco, R. C. and J. Oncina: 1998, 'Learning deterministic regular grammars from stochastic samples in polynomial time'. *RAIRO (Theoretical Informatics and Applications)* **to appear**.

Cover, T. M. and J. A. Thomas: 1991, *Elements of Information Theory*, Wiley Series in Telecommunications. New York, NY, USA: John Wiley & Sons.

Feller, W.: 1950, *An Introduction to Probability Theory and Its Applications I.* New York: John Wiley, second edition.

Gécseg, F. and M. Steinby: 1984, *Tree Automata.* Budapest: Akadémiai Kiadó.

Hoeffding, W.: 1963, 'Probability inequalities for sums of bounded random variables'. *Journal of the American Statistical Association* **58**(301), 13–30.

Hopcroft, J. and J. Ullman: 1980, *Introduction to Automata Theory, Languages, and Computation*. N. Reading, MA: Addison-Wesley.

Oncina, J. and P. García: 1994, 'Inference of rational tree sets'. Technical Report DSIC-ii-1994-23, DSIC, Universidad Politécnica de Valencia.

Sakakibara, Y.: 1992, 'Efficient Learning of Context-Free Grammars from Positive Structural Examples'. *Information and Computation* **97**(1), 23–60.

Sakakibara, Y., M. Brown, R. C. Underwood, I. S. Mian, and D. Haussler: 1994, 'Stochastic Context-Free Grammars for Modeling RNA'. In: L. Hunter (ed.): *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 5 : Biotechnology Computing*. Los Alamitos, CA, USA, pp. 284–294.

Stolcke, A. and S. Omohundro: 1993, 'Hidden Markov Model Induction by Bayesian Model Merging'. In: S. J. Hanson, J. D. Cowan, and C. L. Giles (eds.): *Advances in Neural Information Processing Systems*, Vol. 5. pp. 11–18.

Wetherell, C. S.: 1980, 'Probabilistic Languages: A Review and Some Open Questions'. *ACM Computing Surveys* **12**(4), 361–379.