Stochastic inference of regular tree languages*

Rafael C. Carrasco, Jose Oncina and Jorge Calera

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante
E-mail: (carrasco, oncina, calera)@dlsi.ua.es

Abstract. We generalize a former algorithm for regular language identification from stochastic samples to the case of tree languages or, equivalently, string languages where structural information is available. We also describe a method to compute efficiently the relative entropy between the target grammar and the inferred one, useful for the evaluation of the inference.

1 Introduction

A common concern in the grammatical inference approach is to avoid overgeneralization. Although complete samples 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. Some algorithms for learning regular (string) languages from stochastic samples have been proposed before (Stolcke & Omohundro, 1993; Carrasco & Oncina, 1994). 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 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 & 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 (and therefore, any backwards-deterministic CFG).

In this paper, we introduce a modification of the last algorithm that can be trained with positive stochastic samples generated according to a probabilistic production scheme. The construction follows the same guidelines as the algorithm for string languages in Carrasco & Oncina (1994). We also describe a method to directly evaluate the relative entropy between the inferred language and the true grammar which avoids the generation of large test sets. The relative entropy measures the distance between languages and is usually approximated by means of numerical estimations over large samples generated with the correct

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

distribution. However, this method is in general unfeasible in the case of tree languages, due to the huge number (enormous compared with the case of strings) of different trees that one has to generate. Therefore, an alternative method for such evaluation is of interest.

2 Regular tree languages

Ordered labeled trees will be represented using the functional notation: for instance, the functional notation for the tree shown in Fig. 1 is $a(b(a(bc))c)$. Given a finite set of labels V, the set of all finite-size trees whose nodes are labeled with symbols in V will be denoted as V^T. Any symbol a in V is also the representation of a tree consisting of a single node labeled with a and, therefore, $V \subset V^T$.

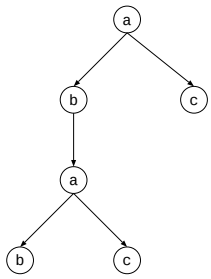

Fig. 1. A graphic representation of the ordered labeled tree $a(b(a(bc))c)$

Deterministic tree automata (DTA) generalize deterministic finite-state automata (DFA), which work on strings. In contrast with DFA's —where strings are processed from left to right—, in DTA's 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 descendents 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, the 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:

$$\delta(t) = \begin{cases} \delta_k(f, \delta(t_1), \ldots, \delta(t_k)) & \text{if } \; t = f(t_1 \ldots t_k) \in (V^T - V) \\ \delta_0(a) & \text{if } \; t = a \in V \end{cases} \tag{1}$$

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 an *absorption* state, i.e., to non-acceptable trees.

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[1]. 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 $q \in Q$,

$$\sum_{f \in V} \sum_{k=0}^{n} \sum_{\substack{q_1,...,q_k \in Q: \\ \delta_k(f,q_1,...,q_k)=q}} p_k(f, q_1, ..., q_k) = 1 \tag{2}$$

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 substitutes, 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 $r(\delta(t))$:

$$p(t|A) = r(\delta(t))\ \pi(t) \tag{3}$$

with $\pi(t)$ recursively given by

$$\pi(f(t_1, \ldots, t_k)) = p_k(f, \delta(t_1), \ldots, \delta(t_k))\ \pi(t_1) \cdots \pi(t_k)\ . \tag{4}$$

Of course, $\pi(a) = p_0(a)$ for $t = a \in V$. The equations (3) and (4) define a probability distribution $p(t|A)$ which is consistent if

$$\sum_{t \in V^T} p(t|A) = 1. \tag{5}$$

The condition of consistency can be written in terms of matrix analysis. Indeed, let us define the expectation elements:

$$\Lambda_{ij} = \sum_{k=1}^{n} \sum_{f \in V} \sum_{\substack{q_1,q_2,...,q_k \in Q: \\ \delta(f,q_1,...,q_k)=j}} p_k(f, q_1, q_2, ..., q_k)(\delta_{iq_1} + \delta_{iq_2} + \cdots + \delta_{iq_k})\ , \tag{6}$$

where δ_{ij} is Kronecker's delta. Consistency, in the sense of Eq. (5), is preserved if the spectral radius of matrix Λ is smaller than one (Wetherell 1980).

[1] This normalization makes the probabilities of all possible expansions of a tree node to add up to one.

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

$$T_1 = T_2 \Leftrightarrow p(t|T_1) = p(t|T_2) \;\; \forall t \in V^T. \tag{7}$$

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 $s\#t$ is obtained by replacing in s the node marked with \$ by a copy of t. For every stochastic tree language T and $t \in V^T$, the *quotient* $t^{-1}T$ is a stochastic language over $V_{\$}^T$ defined through the probabilities

$$p(s|t^{-1}T) = \frac{p(s\#t|T)}{p(V_{\$}^T\#t|T)} \; . \tag{8}$$

In case $s \notin V_{\$}^T$ then $p(s|t^{-1}T) = 0$. On the other hand, if $p(V_{\$}^T\#t|T) = 0$, the quotient (8) is undefined and we will write $t^{-1}T = \emptyset$.

The Myhill-Nerode's theorem for rational languages (see Hopcroft & Ullman 1979) can be generalized for stochastic rational tree languages. If T is a stochastic RTL, the number of different sets $t^{-1}T$ is finite and a deterministic tree automaton (DTA) accepting $\{t \in V^T : p(t|T) > 0\}$ can be defined. We will call it the *canonical acceptor* $M = (Q^M, V, \delta^M, F^M)$, with:

$$\begin{aligned} Q^M &= \{t^{-1}T \neq \emptyset : t \in V^T\} \\ F^M &= \{t^{-1}T : p(t|T) > 0\} \\ \delta^M(f, t_1^{-1}T, \ldots, t_k^{-1}T) &= f(t_1, \ldots, t_k)^{-1}T \end{aligned} \tag{9}$$

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 $X \subset V^T$, $c_n(X) = \sum_{t \in X} c_n(t)$. Provided that the structure (states and transition functions) of M is known, we can estimate the probability functions in the stochastic DTA from the examples in S_n:

$$r(t^{-1}T) = \frac{c_n(\Delta_{f(t_1,\ldots,t_k)})}{n} \tag{10}$$

$$p_k(f, t_1^{-1}T, \ldots, t_k^{-1}T) = \frac{c_n(V_{\$}^T\#f(t_1, ..., t_k))}{c_n(V_{\$}^T\#\Delta_{f(t_1,\ldots,t_k)})} \; . \tag{11}$$

where $\Delta_t = \{s \in V^T : \delta^M(s) = t^{-1}T)$.

4 Inference algorithm

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

```
algorithm tlips
input: A ⊂ Sub(T) such that K(T) ⊂ A
output: SSub (short subtree set)
        F (frontier set)
begin algorithm
  SSub = F = ∅
  W = V_0 ∩ A
  do ( while W ≠ ∅ )
     x = min W
     W = W − {x}
     if ∃y ∈ SSub : equiv_T(x, y) then
        F = F ∪ {x}
     else
        SSub = SSub ∪ {x}
        W = W ∪ {f(t_1, ..., t_k) ∈ A : t_1, ..., t_k ∈ SSub}
     endif
  end do
end algorithm
```

Fig. 2. Algorithm `tlips`.

The *subtree set* and the *short-subtree set* are respectively defined as

$$\begin{aligned} \mathrm{Sub}(T) &= \{t \in V^T : t^{-1}T \neq \emptyset\} \\ \mathrm{SSub}(T) &= \{t \in \mathrm{Sub}(T) : s^{-1}T = t^{-1}T \Rightarrow s \geq t\} \end{aligned} \tag{12}$$

The *kernel* and the *frontier set* are defined as:

$$\begin{aligned} K(T) &= \{f(t_1, ..., t_k) \in \mathrm{Sub}(T) : t_1, ..., t_k \in \mathrm{SSub}(T)\} \\ F(T) &= K(T) - \mathrm{SSub}(T) \end{aligned} \tag{13}$$

Note that there is exactly one tree in $\mathrm{SSub}(T)$ for every state in Q^M of the canonical acceptor, while the trees in $K(T) - V_0$ correspond to the rules in the generating grammar and, therefore, both $\mathrm{SSub}(T)$ and $K(T)$ are finite.

Finally, we define a boolean function $\mathtt{equiv}_\mathtt{T} : K(T) \times K(T) \to \{\mathrm{true}, \mathrm{false}\}$ such that

$$\mathtt{equiv}_\mathtt{T}(t_1, t_2) = \mathrm{true} \Leftrightarrow t_1^{-1}T = t_2^{-1}T. \tag{14}$$

The following theorems support the inference algorithm:

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

$$\begin{aligned} Q &= \mathrm{SSub}(T) \\ \delta(f, t_1, ..., t_k) &= t \end{aligned} \tag{15}$$

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

```
algorithm comp_n
input: x, y ∈ V^T, S_n
output: boolean
begin algorithm
  do ( ∀t, z : depth_t($) = 1 ∧ (t#z#x ∨ t#z#y) ∈ Sub(S_n) )
     if different(c_n(V_$^T#t#z#x), c_n(V_$^T#x), c_n(V_$^T#t#z#y), c_n(V_$^T#y), α) then
        return FALSE
     endif
  end do
  return TRUE
end algorithm
```

Fig. 3. Algorithm $\texttt{comp}_\texttt{n}$.

Theorem 2. *The algorithm in Fig. 2 outputs* $\mathrm{SSub}(T)$ *and* $F(T)$ *with input* $\texttt{equiv}_\texttt{T}$ *plus any* $A \subset \mathrm{Sub}(T)$ *such that* $K(T) \subset A$.

The proofs can be found in the Appendix. 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 $\texttt{equiv}_\texttt{T}$ out of its domain $K(T)$ and 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 $\texttt{equiv}_\texttt{T}$.

5 Probabilistic inference

In practice, the unknown language T is replaced by the stochastic sample S and the equivalence test $\texttt{equiv}_\texttt{T}(x, y)$ is performed through a probabilistic function $\texttt{comp}_\texttt{n}(x, y)$ 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 $\texttt{comp}_\texttt{n}$ tends to $\texttt{equiv}_\texttt{T}$ when n grows.

According to (14), $\texttt{equiv}_\texttt{T}(x, y) =$ true means $x^{-1}T = y^{-1}T$. This can be checked by means of Eq. (8), but we rather check the conditional probabilities:

$$\frac{p(V_\$^T \# t \# z \# x)}{p(V_\$^T \# z \# x)} = \frac{p(V_\$^T \# t \# z \# y)}{p(V_\$^T \# z \# y)} \tag{16}$$

for all $z \in V_\$^T$ and for all $t \in V_\$^T$ such that \$ is at depth one in t.

In order to check (16) a statistical test is applied to the difference (provided that $t\#z\#x$ or $t\#z\#y$ are in $\mathrm{Sub}(S_n)$):

$$\frac{c_n(V_\$^T \# t \# z \# x)}{c_n(V_\$^T \# z \# x)} - \frac{c_n(V_\$^T \# t \# z \# y)}{c_n(V_\$^T \# z \# y)}. \tag{17}$$

We have chosen a Hoeffding (1963) type test, as described in Fig. 4. This check provides the correct answer with probability greater than $(1 - \alpha)^2$, α being

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

Fig. 4. Algorithm `different`.

an arbitrarily small positive number. Therefore, the algorithm $\mathtt{comp_n}$ plotted in Fig. 3 returns the correct value with probability greater than $(1-\alpha)^{2r}$, where r is smaller than the number of different subtrees in S_n. Because r grows slowly with n, we allow α to depend on r. Indeed, if α decreases faster than $1/r$ then $(1-\alpha)^r$ tends to zero and $\mathtt{comp_n}(x,y) = \mathtt{equiv_T}(x,y)$ in the limit of large n. Finally, note that the complexity of $\mathtt{comp_n}$ is at most $\mathrm{O}(n)$. As $|K(T)|$ does not depend on S_n then the global complexity of our algorithm is $\mathrm{O}(n)$.

6 Relative entropy between stochastic languages

The entropy of a probability distribution $p(t|A)$ over V^T,

$$H(A) = -\sum_{t\in V^T} p(t|A)\log_2 p(t|A)\ , \tag{18}$$

bounds (within a deviation of one bit, see Cover & Thomas 1991) the average length of the string needed to code a tree in V^T provided that an optimal coding scheme is used. Optimal coding implies an accurate knowledge of the source A. If only an approximate model A' is available, the average length becomes:

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

The difference $H(A,A') = G(A,A') - H(A)$ is known as *relative entropy* between A and A' or *Kullback-Leibler distance*, a magnitude which is always a positive number: indeed, a suboptimal coding leads to larger average lengths. Note that $H(A) = G(A,A)$ and, thus, $H(A,A') = G(A,A') - G(A,A)$ and, therefore, a procedure to compute $G(A,A')$ can also be used to compute the entropy of a regular tree language or the relative entropy between two languages.

Recall from Eq. (3) 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 $\log_2 p(t|A') = \log_2 r'(\delta'(t)) + \log_2 \pi'(t)$. On the other hand, the class of subsets $L_{ij} = \{t \in V^T : \delta(t) = i \wedge \delta'(t) = j\}$ for $i \in Q$ and $j \in Q'$ defines a partition in V^T. This allows one to write the contribution to $G(A,A')$ of the

r-terms as

$$G_r(A,A') = -\sum_{\substack{i\in Q\\ j\in Q'}}\sum_{t\in L_{ij}} p(t|A)\log_2 r'(j) = -\sum_{\substack{i\in Q\\ j\in Q'}} r(i)\eta_{ij}\log_2 r'(j) \qquad (20)$$

where η_{ij}, defined as

$$\eta_{ij} = \sum_{t\in L_{ij}} \pi(t), \qquad (21)$$

represents the probability that a node of type $i \in Q$ expands as a subtree t such that $\delta'(t) = j$. It is not difficult to show (Calera & Carrasco 1998) that all η_{ij} can be easily obtained by means of an iterative procedure:

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

$$p_k(f,i_1,i_2,\ldots,i_k)\ \eta_{i_1j_1}^{[t]}\eta_{i_2j_2}^{[t]}\cdots\eta_{i_kj_k}^{[t]} \qquad (22)$$

with $\eta_{ij}^{[0]} = 0$. The iterative series monotonically converges to the correct values, as it can be proved straightforwardly by induction.

In order to evaluate the contribution to $G(A,A')$ of the π-terms,

$$G_\pi(A,A') = -\sum_{t\in V^T} p(t|A)\log_2 \pi'(t)\ , \qquad (23)$$

recall that

$$\log_2 \pi'(f(t_1,\ldots t_k)) = \log_2 p'_k(f,\delta'(t_1),\ldots,\delta'(t_k)) + \log_2 \pi'(t_1) + \cdots + \log_2 \pi'(t_k), \qquad (24)$$

so that the contribution of the π-terms becomes

$$G_\pi(A,A') = -\sum_{k=0}^{n}\sum_{f\in V}\sum_{j_1,\ldots,j_k\in Q'} \log_2 p'_k(f,j_1,\ldots,j_k)\ n'(f,j_1,\ldots,j_k) \qquad (25)$$

where $n'(f,j_1,j_2,\ldots,j_k)$ is the expected —according to the distribution $p(t|A)$— number of subtrees $f(t_1,t_2,\ldots,t_k)$ in t such that $\delta'(t_1) = j_1$, $\delta'(t_2) = j_2$,..., $\delta'(t_k) = j_k$. This leads to

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

$$p_k(f,i_1,i_2,\ldots,i_k)\log_2 p'_k(f,j_1,j_2,\ldots,j_k)\eta_{i_1j_1}\eta_{i_2j_2}\cdots\eta_{i_kj_k} \qquad (26)$$

where C_q is the expectation number of subtrees of type q. This vector $\mathbf{C}$ of expectation values C_i can be easily computed using the matrix Λ defined in Eq. (6) together with 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:

$$C_i^{[t+1]} = r(i) + \sum_{j \in Q} \Lambda_{ij} C_j^{[t]} \tag{27}$$

with $C_i^{[0]} = 0$. As in the case of Eq. (22), it is straightforward to show that the iterative procedure converges monotonically to the correct value.

7 An example

The following probabilistic context-free grammar generates conditional statements:

statement → **if** *expression* **then** *statement* **else** *statement* **endif**	(0.2)
statement → **if** *expression* **then** *statement* **endif**	(0.4)
statement → **print** *expression*	(0.4)
expression → *expression* **operator** *term*	(0.5)
expression → *term*	(0.5)
term → **number**	(1.0)

where variables appear in italics, terminals in bold and the number in parenthesis represents the probability of the rule. The average number of 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 150 examples) the correct grammar is always found. Similar behavior was observed for other grammars and experiments. On the other hand, 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 section 6. 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 at a much higher rate than a mere estimation of the probabilities from the sample.

8 Conclusions

The algorithm `tlips` learns context-free grammars from stochastic examples of parse tree skeletons. The result is, in the limit, structurally identical to the target grammar (i.e., they generate the same stochastic set of skeletons) and is

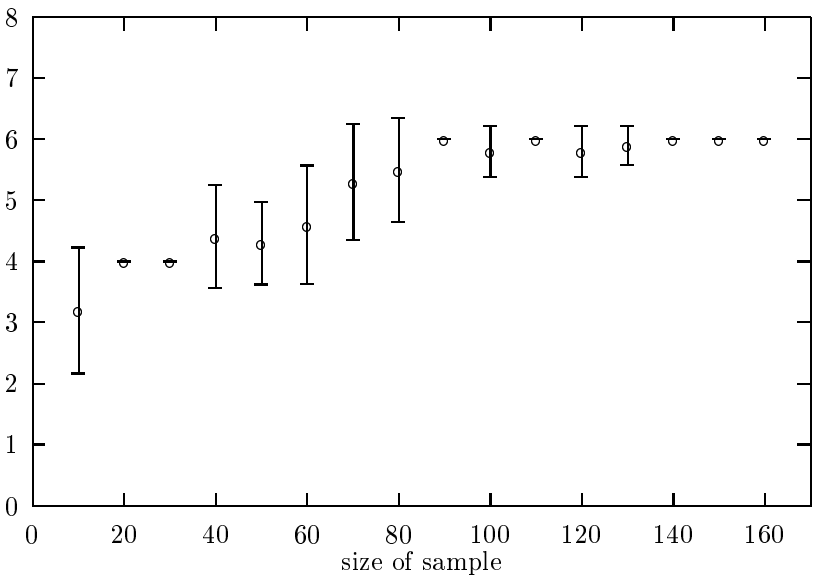

Fig. 5. Average number of rules in the hypothesis as a function of the number of examples. The target grammar has 6 rules.

found in linear time with the size of the sample. Experimentally, identification 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.

A Proof of theorems.

Theorem 1: Let Φ be a mapping that for every language $t^{-1}T$ gives the only tree $s = \Phi(t^{-1}T)$ in $\mathrm{SSub}(T)$ such that $s^{-1}T = t^{-1}T$. Clearly, if $t \in \mathrm{SSub}(T)$ then $\Phi(t^{-1}T) = t$. The mapping Φ is an isomorphism if

$$\delta(f, \Phi(t_1^{-1}T), ..., \Phi(t_k^{-1}T)) = \Phi\delta^M(f, t_1^{-1}T, ..., t_k^{-1}T).$$

As $t_1, ..., t_k$ are in $\mathrm{SSub}(T)$, and $\delta^M(f, t_1^{-1}T, ..., t_k^{-1}T) = f(t_1, ..., t_k)^{-1}T$, then one can rewrite the above condition as

$$\delta(f, t_1, ..., t_k) = \Phi(f(t_1, ..., t_k)^{-1}T)$$

which holds if $\delta(f, t_1, ..., t_k)$ is the only tree $t \in \mathrm{SSub}(T)$ satisfying $t^{-1}T = f(t_1, ..., t_k)^{-1}T$. Note that $t_1, ..., t_k \in \mathrm{SSub}(T)$ implies $f(t_1, ..., t_k) \in K(T)$, and therefore, the condition can be written as $\mathtt{equiv_T}(t, f(t_1, ..., t_k))$.

Theorem 2 (sketch): Simple induction 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.

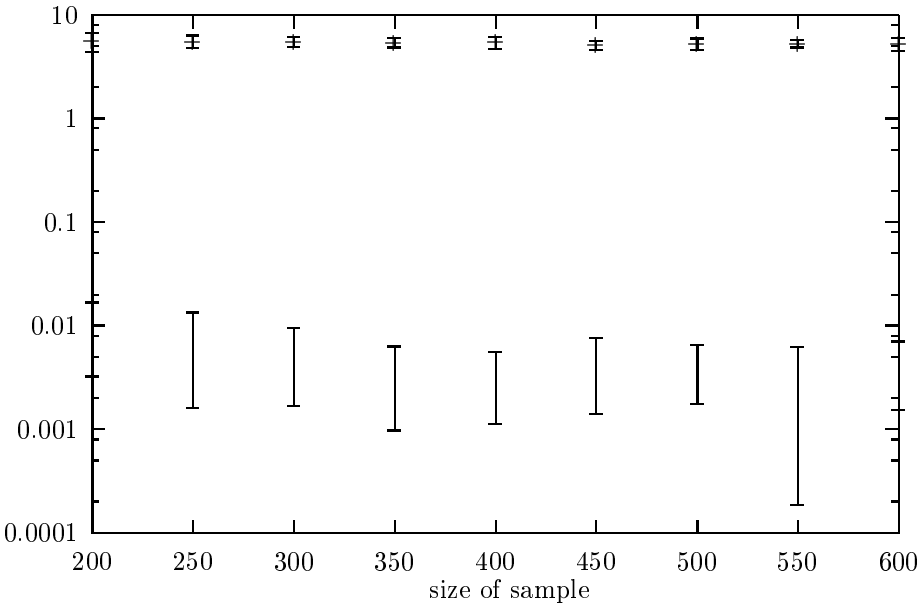

Fig. 6. Lower dots: relative entropy between the target grammar and the output of algorithm `tlips` as a function of the number of examples in the sample. Upper dots: relative entropy between the target grammar and the sample.

References

- Aho, A.V. & Ullman, J.D. (1972): "The theory of parsing, translation and compiling. Volume I: Parsing". Prentice-Hall, Englewood Cliffs, NJ.
- Angluin, D. (1982): Inference of reversible languages. *Journal of the Association for Computing Machines* **29**, 741–765.
- Calera, J. & Carrasco, R.C: (1998): Computing the relative entropy between regular tree languages. Submitted for publication.
- Carrasco, R.C. (1997): "Inferencia de lenguajes racionales estocásticos". Ph.D. dissertation. Universidad de Alicante.
- Carrasco, R.C. (1998): Accurate computation of the relative entropy between stochastic regular grammars. *Theoretical Informatics and Applications*. To appear.
- Carrasco, R.C. & Oncina, J. (1994): Learning stochastic regular grammars by means of a state merging method in "Grammatical Inference and Applications" (R.C. Carrasco and J. Oncina, Eds.). Lecture Notes in Artificial Intelligence **862**, Springer-Verlag, Berlin.
- Cover, T.M & Thomas, J.A. (1991): *Elements of Information Theory*. John Wiley and Sons, New York.
- Hoeffding, W. (1963): Probability inequalities for sums of bounded random variables. *American Statistical Association Journal* **58**, 13–30.
- Hopcroft, J.E. & Ullman, J.D. (1979): "Introduction to automata theory, languages and computation". Addison Wesley, Reading, Massachusetts.

– Oncina, J. & García, P. (1994): Inference of rational tree sets. Universidad Politécnica de Valencia, Internal Report DSIC-ii-1994-23.
– Sakakibara, Y. (1992): Efficient learning of context-free grammars from positive structural examples. *Information and Computation* **97**, 23–60.
– Stolcke, A. & Omohundro, S. (1993): Hidden Markov model induction by Bayesian model merging in "Advances in Neural Information Processing Systems **5**" (C.L. Giles, S.J. Hanson and J.D. Cowan, Eds.). Morgan-Kaufman, Menlo Park, California.
– Wetherell, C.S. (1980): Probabilistic Languages: A Review and Some Open Questions *ACM Computing Survey* 12 361–379