TREE LANGUAGES ARITHMETIC COMPRESSION

Jorge Calera-Rubio, Rafael C. Carrasco and Jose Oncina *

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

Abstract

In this paper, we explore the applicability to compression tasks of the algorithms for regular language inference from stochastic samples. We compare two arithmetic encoders based upon two different kinds of formal languages: string languages and tree languages. The experiments show that tree-based methods outperform the predictive capability of string-based methods when they are applied to files containing structural information and, then, they allow for better compression rates.

Keywords: Formal languages; automata theory; inductive learning; arithmetic compression.

1 Introduction

Stochastic samples are collections of examples that have been generated following a probabilistic distribution. There are different reasons that make inferring languages from stochastic samples an interesting issue:

1. Inference methods usually tend to overgeneralize the input data. There exist different ways to overcome this bias: for instance, one can assume that the examples are randomly generated from a given unknown stationary source. This assumption allows one to avoid using counter-examples which are usually scarce or not representative. For instance, not all sounds which are not an "a" can appear in speech. It is not unusual, however, that the examples used to learn a language come from a random or noisy source.

2. Improving the results in prediction or data compression tasks requires accurate stochastic models of the source generating the data.

On the other hand, there exist situations (as handwritten character recognition) where the string representation does not capture the richness of the input. Indeed, other techniques, as using trees to represent the inputs, are more adequate because they allow one to describe hierarchical relations between the components and they incorporate in a natural fashion structural information, i.e., information about how the representation was generated. Another point worth to be remarked is that any method used to identify regular tree languages can be used to identify context-free languages if the samples contain structural information [13].

In this paper we describe, in section 2, how to integrate both requirements (stochastic identification and tree description) and explore, in section 3, its applicability to information compression in tasks where the data are hierarchically structured.

*The authors thank the Spanish CICYT for partial support through project TIC97-0941.

```
algorithm 1
input: S (sample set)
output: Ω(Σ*) (set of strings)
Ω(Σ*) = ∅
x_1 = λ, m = 1
do  while i ≤ m
    if Ω(x_i) = x_i  then
       add x_i to Ω(Σ*)
       for j = 1 to |Σ|
           if x_i a_j is a prefix in S then
              m = m + 1, x_m = x_i a_j
           endif
       endfor
    endif
    i = i + 1
end do
```

Figure 1: Algorithm computing $\Omega(\Sigma^*)$.

2 Identifying stochastic regular languages

In previous work, we developed a collection of methods to identify regular languages, both for string representations [4] and tree representations [3], when samples are stochastically generated. They are briefly described in this section.

2.1 String languages

In the case of string regular languages, our method is an extension of the method described by Lang [9] for deterministic finite-state automata (DFA) based, in turn, in a previous one by Trakhtenbrot and Barzdin [12] and extended by Oncina, García and Vidal [11] to a more general class of finite state machines (Mealy extended machines that perform translation tasks).

Given the *alphabet* $\Sigma = \{a_1, a_2, ..., a_{|\Sigma|}\}$, we denote with Σ^* the *universal language* of strings obtained by concatenation of symbols in Σ. The special symbol λ represents a string of length zero (the empty string) and # denotes the end of string. The *canonical order* is the relation order in Σ^* such that $x < y$ means one of the following: either x is shorter than y or both have same length and x precedes y alphabetically. Given any DFA [8] $M = (Q, \Sigma, \delta, q_I, F)$, it is possible to identify every node $q \in Q$ in the automaton with a single string in Σ^*: the first string (following canonical order) leading from the initial state q_I to node q. In particular, for $w \in \Sigma^*$, let $\Omega(w)$ be the string characterizing node $\delta(q_I, w)$. It is straightforward to prove that the structure of the DFA is completely defined once function Ω is known, as the set $\Omega(\Sigma^*)$ is isomorphic to Q [4] and the transition function δ is then defined by $\delta(w, a) = \Omega(wa)$.

In order to evaluate $\Omega(\Sigma^*)$, one can use the algorithm in Fig. 1. Note that there is an implicit loop when checking $\Omega(x_i) = x_i$ whose index j ranges from 1 to $i - 1$ and looks for the first string x_j equivalent to x_i, that is, looks for $x_j < x_i$ such that $x_j = \Omega(x_i)$. Stochastic regular languages are defined by a stochastic DFA $M = (Q, \Sigma, \delta, p)$ where for every transition $\delta(q_i, a)$ the automaton also includes a transition probability $p(q_i, a)$ normalized in such way that all transition probabilities with the same starting node (end of string symbol included) sum up to one:

$$\sum_{a \in \Sigma \cup \{\#\}} p(q_i, a) = 1 \tag{1}$$

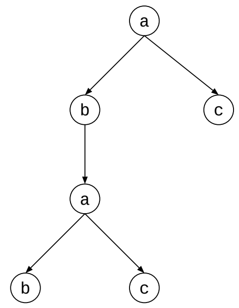

Figure 2: Labelled tree.

Function Ω can also be defined for stochastic automata in the following way: let x_j be a candidate string for $\Omega(x_i)$. Then, for all symbols a in the alphabet, the probability $P(x_i a\Sigma^*|x_i\Sigma^*)$ of the strings starting with $x_i a$ conditioned to the fact that they contain the prefix x_i must coincide with the probability $P(x_j a\Sigma^*|x_j\Sigma^*)$ of the strings starting with $x_j a$ conditioned to the prefix x_j; moreover, $\Omega(x_i a) = \Omega(x_j a)$.

Using the former definition, it is possible to obtain from an experimental sample S an approximate function which approaches Ω with increasing accuracy as the number n of examples in S increases. For this purpose, it is enough to apply a stochastic test to the experimental frequencies (in practice, the true probabilities are unknown) whose confidence level $1 - \alpha_n$ depends on the size of the sample in a way such that $\sum_n n\alpha_n < \infty$. This is a consequence of the Borel-Cantelli lemma [7], as the expected number of stochastic checks that the algorithm performs cannot grow faster than linearly with the sample size (note that the recursion in Ω stops if the string is not a prefix of the sample).

2.2 Tree languages

With the given alphabet Σ it is possible to build objects with a richer structure than strings: for instance, labelled trees. We will denote with Σ^T the set of all different labelled trees that can be build using the symbols in Σ for the node labels. These trees can be coded using *functional notation*: every node is represented as a function whose name is the node label and whose arguments are the subtrees generated by the siblings of the node. For instance, the tree in fig. 2 is described in functional notation as $a(b(a(bc))c)$. Our algorithm needs to define an order relation between trees consistent with the depth of the trees[1], that is, shallow trees must precede deeper trees.

Regular tree languages are those languages that can be recognized by deterministic tree automata (DTA). A DTA is a generalization of a DFA and is able to process (bottom-up) labelled trees: a state is assigned to each node in the tree depending on its label and depending also on the states tied to the siblings of the node. In a way parallel to DFA, the analysis ends when the last position is reached (in this case, the root of the tree) and the type of node (accepting or non-accepting) defines the output of the automaton. Note that it is not necessary to introduce an initial state as the analysis starts simultaneously from all leaves and only their label determines the state tied to the leaf.

In order to use a DTA as a stochastic model, it is necessary to assign a probability to every state transition and also to every node q. Formally, a stochastic DTA is $A = (Q, \Sigma, \delta, p, r)$, where

- $Q = \{q_1, q_2, ..., q_{|Q|}\}$ is a finite set of states;
- $\Sigma = \{a_1, a_2, ..., a_{|\Sigma|}\}$ is the finite set of labels;

[1] The depth of a tree is given by the maximum distance between a leaf and the root of the tree.

```
algorithm 2
input: S (sample set)
output: Ω(Σ^T) (set of subtrees)
Ω(Σ^T) = ∅  for i = 1 to |Σ|
                x_i = a_i
endfor
m = |Σ|
do  while i ≤ m
    if Ω(x_i) = x_i  then
        add x_i to Ω(Σ^T)
        do  ∀t = f(t_1, ...t_k) subtree in S not in {x_1, ..., x_m}
            if (t_1, ..., t_k) ∈ Ω(Σ^T)^k then
                m = m + 1, x_m = t
            endif
        end do
    endif
end do
```

Figure 3: Algorithm computing $\Omega(\Sigma^T)$.

- $\delta = \{\delta_0, \delta_1, ...\delta_n\}$ is a set of transition functions;
- $p = \{p_0, p_1, ...p_n\}$ is the set of transition probabilities;
- $r : Q \to [0, 1]$ is the probability that the tree is of type q.

The last function satisfies $\sum_{q \in Q} r(q) = 1$. Because the number of siblings of a node is not fixed, we need a set of transition functions (rather than a single one) $\delta = \{\delta_0, \delta_1, ...\delta_n\}$, where n is the maximum number of siblings allowed in the language. Every function δ_k takes as arguments a symbol in Σ and k states (one for every sibling of the node) and returns a new state, that is, $\delta_k : \Sigma \times Q^k \to Q$.

For instance, if t is a leaf subtree with label a, then $k = 0$ and the state tied to t is $\delta(t) = \delta_0(a)$. However, if t is a subtree labelled f with two siblings generating subtrees t_1 and t_2 respectively, then $t = f(t_1, t_2)$ and the state tied to t is $\delta(t) = \delta_2(f, \delta(t_1), \delta(t_2))$. By convention, undefined transitions lead to invalid trees.

The normalizing condition for the set of functions $p_k : \Sigma \times Q^k \to [0, 1]$ is that all probabilities of transitions leading to the same state q must sum up to one. That is, for all $q \in Q$

$$\sum_{f \in \Sigma} \sum_{k=0}^{n} \sum_{\substack{q_1, q_2, ..., q_k \in Q \\ q = \delta(f, q_1, q_2, ..., q_k)}} p_k(f, q_1, ..., q_k) = 1. \tag{2}$$

Once the functions p_k are given, the probability that a tree t is generated is the product of all transition probabilities used while analyzing t. In order to ensure that the sum of probabilities for all trees is one, this result has to be multiplied by $r(q)$, being $q = \delta(t)$ the state tied t, that is, $p(t|A) = r(\delta(t))\ \pi(t|\delta(t))$, where $\pi(t|\delta(t))$ represents the probability that t is generated from state $q = \delta(t)$, a number that can be recursively computed. For instance, for a tree $t = f(t_1, t_2)$, one gets:

$$\pi(t|\delta(t)) = p_2(f, \delta(t_1), \delta(t_2))\ \pi(t_1|\delta(t_1))\ \pi(t_2|\delta(t_2)). \tag{3}$$

In a way similar to stochastic DFA, the structure of the stochastic DTA is defined (see algorithm 2) by a function $\Omega(t) = s$ giving the first tree s such that $\delta(s) = \delta(t)$. Searching for a tree x_j candidate to $\Omega(x_i)$ requires the following checks:

1. The relative frequency in the sample of trees with root type x_j must be similar to the relative frequency of trees with root type x_i.

2. The relative frequency in a given context $t_j = f(s_1, ..., x_j, ..., s_k)$ of x_j-type nodes must be similar to the relative frequency of x_i-type nodes in the same context $t_i = f(s_1, ..., x_i, ..., s_k)$; moreover $\Omega(t_i) = \Omega(t_j)$.

The meaning of the word similar in the former paragraph is given by a statistical check applied to the experimental frequencies. Once more, if we choose a the confidence level $1 - \alpha_n$ satisfying $\sum_n n\alpha_n < \infty$, the number of mistakes as n (the sample size) grows is finite.

In this work, we have improved the inference algorithms described in this section in order to reuse the information carried by the states which are found to be equivalent to another one. Note that in the previous description we neglect the information about the strings (or trees) containing a prefix (subtree) x_j equivalent to a previous one x_i. However, this information can speed up the convergence of the identification process. Therefore, we have introduced a state merging technique that improves the efficiency although makes implementation considerably harder.

3 Application of the stochastic models to tree compression

We implemented two arithmetic encoders[6, 10] and their corresponding decoders based upon the inference methods described in former section and applied them to files containing data structured as trees. For this purpose we used the following tree grammar:

$$
\begin{array}{rlcll|rlcll}
1: & q_0 & = & \delta_7(S, q_4, q_1, q_5, q_0, q_6, q_0, q_7) & (0.2) & 10: & q_4 & = & \delta_0(\mathtt{i}) & (1.0) \\
2: & q_0 & = & \delta_5(S, q_4, q_1, q_5, q_0, q_7) & (0.2) & 11: & q_5 & = & \delta_0(\mathtt{t}) & (1.0) \\
3: & q_0 & = & \delta_2(S, q_8, q_1) & (0.6) & 12: & q_6 & = & \delta_0(\mathtt{e}) & (1.0) \\
4: & q_1 & = & \delta_3(E, q_1, q_9, q_2) & (0.3) & 13: & q_7 & = & \delta_0(\mathtt{f}) & (1.0) \\
5: & q_1 & = & \delta_1(E, q_2) & (0.7) & 14: & q_8 & = & \delta_0(\mathtt{p}) & (1.0) \\
6: & q_2 & = & \delta_3(T, q_2, q_{10}, q_3) & (0.2) & 15: & q_9 & = & \delta_0(\mathtt{+}) & (1.0) \\
7: & q_2 & = & \delta_1(T, q_3) & (0.8) & 16: & q_{10} & = & \delta_0(\mathtt{*}) & (1.0) \\
8: & q_3 & = & \delta_1(F, q_{11}) & (0.9) & 17: & q_{11} & = & \delta_0(\mathtt{n}) & (1.0) \\
9: & q_3 & = & \delta_1(F, q_1) & (0.1) & & & & &
\end{array} \quad (4)
$$

The first number identifies the rule, the number in parenthesis is the transition probability and terminals appear in typewriter font. Moreover, $r(q_i)$ is one if $i = 0$ and zero otherwise. In this way, the sentences that can be generated are parse trees of conditional structures such as `if...then...else...endif`, `if...then...endif` and language commands as `print` together with numerical expressions (`n`) linked with sum or product operators. For instance, the sentence "`i n t p n e p n f`" has a derivation tree:

$$S(\mathtt{i}E(T(F(\mathtt{n})))\mathtt{t}S(\mathtt{p}E(T(F(\mathtt{n}))))\mathtt{e}S(\mathtt{p}E(T(F(\mathtt{n}))))\mathtt{f})$$

An alternative representation for this tree is the string of rules applied in its (rightmost) derivation:

$$1; 13; 3; 5; 7; 8; 17; 14; 12; 3; 5; 7; 8; 17; 14; 11; 5; 7; 8; 17; 10$$

where each number identifies one rule in the grammar. Of course, in this representation, the probability of the tree is easily obtained by simply multiplying the rule probabilities. In the string arithmetic encoder (decoder), the input is encoded (decoded) according to the probabilities the model predicts for every continuation after a prefix. In the case of trees, our model predicts what rules can be applied at a given point in the rightmost derivation and generates the codes according to this probability distribution (except for the very the first symbol, which is predicted

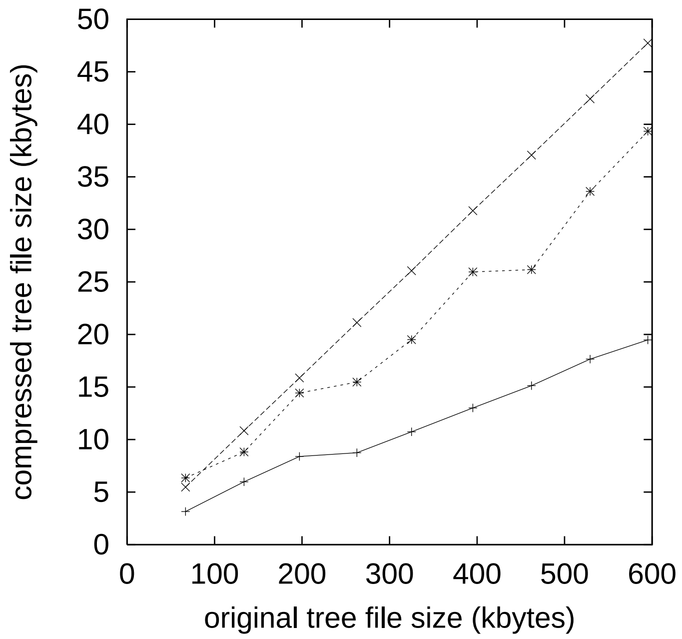

Figure 4: Compressed file size. Long dashed: `gzip`; short dashed: string arithmetic encoding; continuous: tree arithmetic encoding.

according to $r(q_i)$). This procedure becomes more efficient when the grammar is $LL(1)$ [1]. In such case, both encoding and decoding can be implemented in a sequential way.

In figure 4, we show the compressed size of the files generated with the example grammar after our arithmetic encoders are applied and compare it to the size obtained with a standard compressor as `GNU's gzip`. The model used for the arithmetic coding is included as a header in the compressed file. As seen in the figure, the string model gives compression rates around 12 that are close to those obtained with `gzip`. However, the tree model reduces the size of the compressed file by a factor of two (between 25 and 30 times the original one). We have checked that in all cases the results are consistent with the entropic lower limit of the compressed file [5, 2]:

$$\sum_{t \in F} \log_2 p(t|A') \tag{5}$$

where F is the sample file and $p(t|A')$ is the probability that our model A' assigns to the tree t. Indeed, all encoders perform close to the bounds given by (5). However, both `gzip` and the string encoder are not able to use the *a priori* knowledge that the tree model incorporates: that is, that the file contains trees. This fact is responsible for the additional compression obtained with the tree model.

Although the size of the model is relatively small for large files, tree inference would be completely useful if an incremental method was used. This would avoid coding the model as part of the compressed file. Finally, it should be remarked that arithmetic encoders could be also used for on-line compression if an *a priori* model is available, while `gzip` cannot compress isolated trees.

4 Conclusions

We have implemented two arithmetic encoders based upon the inference models we developed previously for stochastic languages. The first one learns from string samples while the second one learns from tree samples. In case the file to be compressed includes structural information (that is, the information is described as trees), the string models provide similar results to the Lempel-Ziv compression methods. However, tree models allow for higher compression rates. Then, an incremental method which avoids coding the model would be of interest.

References

[1] Aho, A.V. & Ullman, J.D. "The theory of parsing, translation and compiling. Volume I: Parsing". Prentice-Hall, Englewood Cliffs, NJ (1972).

[2] Calera-Rubio, J. & Carrasco, R.C. "Computing the relative entropy between regular tree languages". Information Proccessing Letter (1998). To appear.

[3] Carrasco, R.C, Oncina, J. y Calera-Rubio, J. "Stochastic inference of regular tree languages". Proceedings of the 3rd International Colloquium on Grammatical Inference. Ames (Iowa). Lecture Notes on Artificial Intelligence (1998) **1433** , 187-198.

[4] R.C. Carrasco and J. Oncina. "Learning deterministic regular grammars from stochastic samples in polynomial time". Theoretical Informatics and Applications (1998). To appear.

[5] Carrasco, R.C. "Accurate computation of the relative entropy between stochastic regular grammars". Theoretical Informatics and Applications **31** (1997), 437-444.

[6] Cover, T.M and Thomas, J.A. "Elements of Information Theory". John Wiley and Sons, New York (1991).

[7] Feller, W. "An introduction to probability theory and its applications". John Wiley and Sons, New York (1950).

[8] Hopcroft, J.E. and Ullman, J.D. "Introduction to automata theory, languages and computation". Addison Wesley, Reading, Massachusetts (1979).

[9] Lang, K. "Random DFA's can be Approximately Learned from Sparse Uniform Examples". Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory (1992).

[10] Nelson, M.R. (1991): "Arithmetic coding and statistical modeling". Dr. Dobb's Journal of Software Tools (1991) **16** (February).

[11] Oncina, J., García, P. y Vidal, E. "Learning subsequential transducers for pattern recognition interpretation tasks". IEEE Transactions on Pattern Analysis and Machine Intelligence (1993) **15** 448-458.

[12] Trakhtenbrot,B.A. and Barzdin, Y.M. "Finite Automata - Behavior and Synthesis". North-Holland, Amsterdam (1973).

[13] Sakakibara, Y. "Efficient learning of context-free grammars from positive structural examples". Information and Computation (1992) **97**,23–60.