Stochastic k-testable Tree Languages and Applications

Juan Ramón Rico-Juan, Jorge Calera-Rubio and Rafael C. Carrasco*

Departament de Llenguatges i Sistemes Informàtics, Universitat d'Alacant, E-03071 Alacant, Spain
{juanra, calera, carrasco}@dlsi.ua.es

Abstract. In this paper, we describe a generalization for tree stochastic languages of the k-gram models. These models are based on the k-testable class, a subclass of the languages recognizable by ascending tree automata. One of the advantages of this approach is that the probabilistic model can be updated in an incremental fashion. Another feature is that backing-off schemes can be defined. As an illustration of their applicability, they have been used to compress tree data files at a better rate than string-based methods.

Keywords: tree grammars, stochastic models, backing-off, data compression

1 Introduction

Stochastic models based on k-grams have been widely used in natural language modeling (Brown et al., 1992; Ney et al., 1995), speech recognition (Jelinek, 1998) and data compression (Rubin, 1976). Indeed, stochastic models can be used to predict the next symbol in a sequence and, therefore, they are a key component of arithmetic data compression algorithms (Witten et al., 1987; Cover and Thomas, 1991). In classification tasks, the need of stochastic models often arises when the Bayes' decision rule for minimum error rate is applied: given a sequence $S = s_1 s_2 \ldots$ of observations, the stochastic model M that maximizes the conditional probability $P(M|S)$ also maximizes $P(S|M)P(M)$ and, therefore, a model $P(S|M)$ for the generation of sequences is needed. If the stochastic model is of the form $P(S = s_1 s_2 \ldots s_t|M) = p_M(s_1)p_M(s_2|s_1)\cdots p_M(s_t|s_1 s_2 \ldots s_{t-1})$ and the conditional probabilities are assumed to depend only on the last $k-1$ words, $p_M(s_t|s_1 \ldots s_{t-1}) = p_M(s_t|s_{t-k+1} \ldots s_{t-1})$, the resulting Markov chain model (Chung, 1967) is known as k-gram model.

From a theoretical point of view, k-gram models can be regarded as a probabilistic extension of locally testable languages (García and Vidal, 1990; Yokomori, 1995). Informally, a string language L is locally testable if every string w can be recognized as a string in L just by looking at all the substrings in w of length at most k.

* Work supported by the Spanish Comisión Interministerial de Ciencia y Tecnología through grant TIC2000-1599-C02.

On the other hand, previous work (Knuutila, 1993; Knuutila and Steinby, 1994; García, 1993) has proposed identification algorithms for locally testable tree languages. Trees are a more natural representation of the input when hierarchical relations are established among the pattern components. In particular, stochastic tree grammars have been widely used to tackle ambiguity in natural language parsing (Charniak, 1993; Stolcke, 1995). In this paper, we explore the applicability of stochastic locally testable models to describe tree data. Some notation is introduced in the following section and a description of the model can be found in section 3. We have checked (section 4) that using these models for adaptive compression of tree data results in an improved performance compared to the traditional string-based arithmetic compression. How a backing-off scheme can be defined for classification tasks is described in section 5.

2 Trees and Tree Automata

Given an *alphabet*, that is, a finite set of symbols $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$, the set Σ^T of Σ-trees is defined as the language generated by the context-free grammar $G = (\Sigma', \{T, F\}, T, R)$, where the alphabet Σ' contains Σ plus the left and right parenthesis and whose set of productions R contains the rules:

- $T \longrightarrow \sigma(F)$ for all $\sigma \in \Sigma$
- $F \longrightarrow \lambda \mid TF$

In this definition, λ represents the empty string. For brevity, we will write σ whenever $\sigma(\lambda)$ is generated by G. The *depth* of a tree t is

$$\operatorname{depth}(t) = \begin{cases} 0 & \text{if } t = \sigma \in \Sigma \\ 1 + \max_{j=1}^{m}\{\operatorname{depth}(t_j)\} & \text{if } t = \sigma(t_1 \ldots t_m) \text{ with } m > 0 \end{cases} \tag{1}$$

For instance, the Σ-tree $a(a(a(ab))b)$ belongs to $\{a, b\}^T$ and its depth is 3. Its graphical representation is depicted in Fig.1.

A *deterministic finite-state tree automaton* (DTA) is defined as a four-tuple $A = (Q, \Sigma, \Delta, F)$, where $Q = \{q_1, \ldots, q_{|Q|}\}$ is a finite set of states, $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is the alphabet, $F \subseteq Q$ is the subset of accepting states and $\Delta = \{\delta_0, \delta_1, \ldots, \delta_M\}$ is a collection of transition functions of the form $\delta_m : \Sigma \times Q^m \to Q$. For all trees $t \in \Sigma^T$, the result $\delta(t) \in Q$ of the operation of A on t is

$$\delta(t) = \begin{cases} \delta_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ \delta_m(\sigma, \delta(t_1), \ldots, \delta(t_m)) & \text{if } t = \sigma(t_1 \ldots t_m) \text{ with } m > 0 \end{cases} \tag{2}$$

The tree language $L(A)$ recognized by the automaton A is the subset of Σ^T

$$L(A) = \{t \in \Sigma^T : \delta(t) \in F\}. \tag{3}$$

Every language that can be recognized by a DTA is called a *rational tree language* (Rozenberg and Salomaa, 1997) and generalize the *ascending tree automata* described in Nivat and Podelski (1997).

For instance, if $\Sigma = \{a, b\}$ and Δ contains the transitions $\delta_0(a) = q_1$, $\delta_0(b) = q_2$, $\delta_2(a, q_1, q_2) = q_2$ and $\delta_1(a, q_2) = q_1$, the result of the operation of A on tree $t = a(a(a(ab))b)$, plotted in Fig.1, is $\delta(t) = \delta_2(a, \delta(a(a(ab))), \delta(b))$. Recursively, one gets $\delta(b) = q_2$ and $\delta(a(a(ab))) = q_1$. Then, $\delta(t) = \delta(a, q_1, q_2) = q_2$. By convention, undefined transitions lead to absorption states, that is, to unaccepted trees. Stochastic tree automata generate a probability distribution over

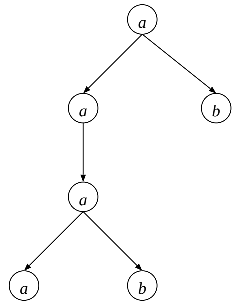

Fig. 1. A graphical representation of the tree $t = a(a(a(ab))b)$.

the trees in Σ^T. A stochastic DTA incorporates a probability for every transition in the automaton, with the normalization that the probabilities of the transitions leading to the same state $q \in Q$ must add up to one. In other words, there is a collection of functions $P = \{p_0, p_1, p_2, \dots, p_M\}$ of the type $p_m : \Sigma \times Q^m \to [0, 1]$ such that they satisfy, for all $q \in Q$,

$$\sum_{\sigma \in \Sigma} \sum_{m=0}^{M} \sum_{\substack{q_1, \dots, q_m \in Q: \\ \delta_m(\sigma, q_1, \dots, q_m) = q}} p_m(\sigma, q_1, \dots, q_k) = 1 \tag{4}$$

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

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

with $\pi(t)$ recursively given by

$$\pi(t) = \begin{cases} p_0(\sigma) & \text{if } t = \sigma \in \Sigma \\ p_m(\sigma, \delta(t_1), \dots, \delta(t_m))\, \pi(t_1) \cdots \pi(t_m) & \text{if } t = \sigma(t_1 \dots t_m) \text{ with } m > 0 \end{cases} \tag{6}$$

[1] We may define accepting states as those $q \in Q$ such that $\rho(q) > 0$

The equations (5) and (6) define a probability distribution $p(t|A)$ which is consistent if

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

As shown in Chaudhuri et al. (1983) and Sánchez and Benedí (1997), context-free grammars whose probabilities are estimated from random samples are always consistent. It is easy to show (Sakakibara, 1992) that identifying a DTA is equivalent to identifying a context-free grammar from structural descriptions. In the following, the probabilities of the DTA will be extracted from random samples and, therefore, consistency is always preserved.

3 Stochastic Extension of Locally Testable Tree Languages

Locally testable languages, in the case of strings, are characterized by defining the set of substrings of length k together with prefixes and suffixes of length strictly smaller than k to check near the string boundaries García and Vidal (1990); Yokomori (1995). In the case of trees, the k-fork plays the role of the substrings and the k-root and k-subtrees are used to check near the tree boundaries. These concepts are formally defined below.

For all $k > 0$ and for all trees $t = \sigma(t_1 \dots t_m) \in \Sigma^T$, the *$k$-root* of t is the tree in Σ^T defined as

$$r_k(\sigma(t_1 \dots t_m)) = \begin{cases} \sigma & \text{if } k = 1 \\ \sigma(r_{k-1}(t_1) \dots r_{k-1}(t_m)) & \text{otherwise} \end{cases} \tag{8}$$

Note that in case $m = 0$, that is $t = \sigma \in \Sigma$, then $r_k(\sigma) = \sigma$.

On the other hand, the set $f_k(t)$ of *k-forks* and the set $s_k(t)$ of *k-subtrees* are defined for all $k > 0$ as follows:

$$f_k(\sigma(t_1 \dots t_m)) = \cup_{j=1}^{m} f_k(t_j) \cup \begin{cases} \emptyset & \text{if } 1 + \text{depth}(\sigma(t_1 \dots t_m)) < k \\ r_k(\sigma(t_1 \dots t_m)) & \text{otherwise} \end{cases} \tag{9}$$

$$s_k(\sigma(t_1 \dots t_m)) = \cup_{j=1}^{m} s_k(t_j) \cup \begin{cases} \sigma(t_1 \dots t_m) & \text{if depth}(\sigma(t_1 \dots t_m)) < k \\ \emptyset & \text{otherwise} \end{cases} \tag{10}$$

In the particular case $t = \sigma \in \Sigma$, then $s_k(t) = f_1(t) = \sigma$ and $f_k(t) = \emptyset$ for all $k > 1$.

For instance, if $t = a(a(a(ab))b)$ then one gets $r_2(t) = \{a(ab)\}$, $f_3(t) = \{a(a(a)b), a(a(a(b)))\}$ and $s_2(t) = \{a(ab), a, b\}$.

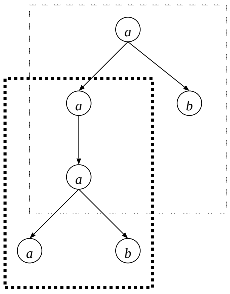

Fig. 2. Set of 3-forks generated in $a(a(a(ab))b)$.

The above definitions allow one to define formally the class of k-testable tree languages. A tree language T is a *strictly k-testable* language (with $k \geq 2$) if there exist three finite subsets $\mathcal{R}, \mathcal{F}, \mathcal{S} \subseteq \Sigma^T$ such that

$$t \in T \Leftrightarrow r_{k-1}(t) \subseteq \mathcal{R} \wedge f_k(t) \subseteq \mathcal{F} \wedge s_{k-1}(t) \subseteq \mathcal{S}. \tag{11}$$

It is straightforward (Knuutila, 1993; Knuutila and Steinby, 1994; García, 1993) to build a DTA $A = (Q, \Sigma, \Delta, F)$ that recognizes a strictly k-testable tree language T. For this purpose, it suffices to choose:

- $Q = \mathcal{R} \cup r_{k-1}(\mathcal{F}) \cup \mathcal{S}$;
- $F = \mathcal{R}$;
- $\delta_m(\sigma, t_1, \ldots, t_m) = \sigma(t_1 \ldots t_m)$ for all $\sigma(t_1 \ldots t_m) \in \mathcal{S}$
- $\delta_m(\sigma, t_1, \ldots, t_m) = r_{k-1}(\sigma(t_1 \ldots t_m))$ for all $\sigma(t_1 \ldots t_m) \in \mathcal{F}$

Furthermore, under the assumption that the tree language L is k-testable, the DTA recognizing L can be identified from positive samples (Knuutila, 1993; Knuutila and Steinby, 1994; García, 1993), that is, from sets made of examples of trees in the language. Given a positive sample S, the procedure to obtain the DTA essentially builds the automaton A using $r_{k-1}(S)$, $f_k(S)$ and $s_{k-1}(S)$ instead of $\mathcal{R}, \mathcal{F}$ and $\mathcal{S}$ respectively in the above definitions for Q, F and Δ.

For instance, for the single example sample $S = \{a(a(a(ab))b)\}$ and $k = 3$ one gets the set of sates $Q = r_2(S) \cup r_2(f_3(S)) \cup s_2(S) = \{a(ab), a(a), a, b\}$ with final subset $F = \{a(ab)\}$ and transitions $\delta_0(a) = a$, $\delta_0(b) = b$ and $\delta_2(a, a, b) = a(ab)$ on the one hand and $\delta_2(a, a(a), b) = a(ab)$ and $\delta_1(a, a(ab)) = a(a)$ on the other hand.

Now we can extend this learning procedure to the case where the sample is stochastically generated. A stochastic sample $S = \{\tau_1, \tau_2, \ldots \tau_{|S|}\}$ consists of a sequence of trees generated according to a given probability distribution. If our model is a stochastic DTA, the distribution $p(t|A)$ is given by equations (5–6). The assumption that the underlying transition scheme (that is, the states Q and the collection of transition functions Δ) correspond to a k-testable DTA allows one to infer a stochastic DTA from a sample in a simple way. For this purpose, one should note that the likelihood of the stochastic sample S is maximized (Ney et al., 1995) if the stochastic model assigns to every tree τ in the sample a probability equal to the relative frequency of τ in S. In other words, every transition in Δ will be assigned a probability which coincides with the relative number of times the rule is used when the trees in the sample are parsed. Therefore, the procedure to infer a stochastic DTA from a stochastic sample $S = \{\tau_1, \tau_2, \dots, \tau_{|S|}\}$ works as follows. The set of states is built as:

$$Q = r_{k-1}(S) \cup r_{k-1}(f_k(S)) \cup s_{k-1}(S); \tag{12}$$

for every $t = \sigma(t_1, \dots, t_m) \in s_{k-1}(S)$, Δ contains a transition

$$\delta(\sigma, t_1, ..., t_m) = \sigma(t_1, ..., t_m); \tag{13}$$

for every $t = \sigma(t_1, \dots, t_m) \in f_k(S)$, Δ contains a transition

$$\delta(\sigma, t_1, ..., t_m) = r_{k-1}(\sigma(t_1, ..., t_m)); \tag{14}$$

for all $t = \sigma(t_1, \dots, t_m) \in s_{k-1}(S) \cup f_k(S)$ the transition probabilities in P are estimated as

$$p_m(\sigma, t_1, \dots, t_m) = \frac{C^{[k]}(\sigma(t_1, \dots, t_m), S)}{C^{[k-1]}(r_{k-1}(\sigma(t_1, \dots, t_m)), S)} \tag{15}$$

where $C(t, S) = \sum_{i=1}^{|S|} C(t, \tau_i)$ and $C(t, \tau_i)$ counts the number of k-forks and $(k-1)$-subtrees[2] isomorphic to t found in τ_i; finally, the subset of accepting states is $F = r_{k-1}(S)$ and the probabilities $\rho(t)$ are estimated for every $t \in F$ as

$$\rho(t) = \frac{1}{|S|} D^{[k]}(t, S) \tag{16}$$

where $D^{[k]}(t, S) = \sum_{i=1}^{|S|} D^{[k]}(t, \tau_i)$ and $D^{[k]}(t, \tau_i) = 1$ if $r_{k-1}(\tau) = t$ and zero otherwise.

It is useful to store the above probabilities as the quotient of two terms, as given by equations (15) and (16). In this way, if a new tree (or subtree) τ is provided, the automaton A can be easily updated to account for the additional information. For this incremental update, it suffices to increment each term in the equations with the partial sums obtained for τ.

4 An application: tree data compression

In this section, we explore the application of the class of models considered here to the task of tree data compression. Because stochastic modeling becomes a key ingredient in arithmetic compression (Witten et al., 1987, 1999), one expects that

[2] Note that a tree τ may contain, depending on the depth of t, either k-forks or $(k-1)$-subtrees isomorphic to t but not both simultaneously.

probabilistic tree models that provide a better description of the file content will allow for a more effective file compression. Conversely, compression performance can be used as a measure of the quality of the stochastic model.

Recall that an arithmetic encoder uses at step n the cumulative range of probabilities $[l(e_n), h(e_n)[$ that the model assigns to the event e_n having probability $h(e_n) - l(e_n)$. Starting with $low_0 = 0$ and $high_0 = 1$, a new interval is iteratively computed as follows

$$\begin{aligned} low_{n+1} &= low_n + (high_n - low_n)l(e_n) \\ high_{n+1} &= low_n + (high_n - low_n)h(e_n) \end{aligned} \tag{17}$$

Implementing this computation using integer arithmetics is a subtle task as shown in Witten et al. (1987). An important issue is that the probabilistic model should never assign a null probability to any event that can be observed, that is, $h(e_n) - l(e_n)$ has to be always strictly positive.

The procedure to compress tree data follows the guidelines of prediction by partial matching (PPM, Cleary and Witten (1984)). Similarly to the case of strings, a probabilistic model $M^{[k]}$ predicts the next code to be transmitted based on the previous context. In this case, the context is given by the $(k-1)$-fork (or when appropriate by the $(k-1)$-root or $(k-1)$-subtree) in the tree above the node. For instance, if $k = 3$, the possible expansions of the nodes shadowed in Fig. 3 depend on the whole context marked with a square. In that case, the probability $p_2(a, a(a), b)$ of the expansion $t = a(a(a)b)$ given the observed state $q = r_2(t) = a(ab)$ is needed. In the following, we will call a tree t expansion if $t \in f_k(S) \cup s_{k-1}(S)$ and root expansion if $t \in r_{k-1}(S)$.

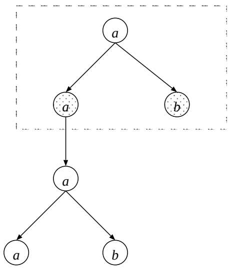

Fig. 3. Trailing context ($k = 3$) for the expansion of the shadowed node.

In case the k-testable model $M^{[k]}$ contains no information about the expansion $t = \sigma(t_1, ..., t_m)$, the $(k-1)$-order model $M^{[k-1]}$ is used instead to compute the probabilities of the subexpansions $t_1, ..., t_m$. This backing-off procedure is repeated recursively till either a) a suitable model is found for the subexpansion under consideration or b) $k = 1$ and the ground model $M^{[1]}$ is applied. On the return, when each state expansion is known, the models $M^{[1]}$ to $M^{[k]}$ are updated.

Some important features differentiate the procedure form the standard string PPM compression and are worth to comment:

1. In contrast to strings, where left-to-right processing is the natural choice, different orders are possible to perform a walk on the tree. Breadth-first traversal offers the advantage that the model can be updated before the whole tree is expanded. This improves compression of files consisting of a single tree or a small amount of them.
2. Each model $M^{[k]}$ with $k > 1$ consists of a collection of counters $C^{[k]}(t)$ and $D^{[k]}(t)$ needed for the estimates (15) and (16) respectively. There is a counter for every different argument t, where t is a k-fork or $(k-1)$-subtree in the first case and a $(k-1)$-root in the second case. In addition, one needs counters for the escape codes: $D^{[k]}(\epsilon_r)$ for the root escape code ϵ_r and $C^{[k]}(\epsilon_q)$ for the escape code ϵ_q associated to the state q. All these counters are initialized to zero, updated with the processed part of the tree and they will be used to estimate the probabilities as follows. The probability in model $M^{[k]}$ of the expansion t from state $q = \sigma(t_1, ..., t_m)$ is given by
$$\alpha(k,t) = \frac{C^{[k]}(t)}{C^{[k-1]}(q) + C^{[k]}(\epsilon_q)} \tag{18}$$
The above formula can also be used for the escape probabilities $\alpha(k, \epsilon_q)$. On the other hand, the probability in $M^{[k]}$ of the root expansion t is
$$\beta(k,t) = \frac{D^{[k]}(t)}{|S| + D^{[k]}(\epsilon_r)} \tag{19}$$
This equation is also holds for the escape code probability $\beta(k, \epsilon_r)$.
3. The ground model $M^{[1]}$ is used when no information is available about a expansion $t = \sigma(a_1, ..., a_m)$ with $a_1, ..., a_m \in \Sigma$ or $t = \sigma$. In such case, $q = r_1(t) = \sigma$ and one needs to code the number m of descendents together with, if $m > 0$, their labels $a_1, ..., a_m$. Therefore, this ground model has two components. On the one hand, a collection of counters $E_\sigma(m)$ stores how many nodes labeled σ expanded m subtrees. As the maximal tree-width M is in advance unknown, we initially use M counters per symbol, $E_\sigma(0), ..., E_\sigma(M-1)$ plus an additional one $E_\sigma(\epsilon_\sigma)$ that stores how often M additional counters are needed. Then, the probability that a node labeled σ has m descendents is computed as follows
$$\gamma(\sigma,m) = \frac{1 + E_\sigma(m)}{1 + M E_\sigma(\epsilon_\sigma) + \Sigma_{i=0}^{M E_\sigma(\epsilon_\sigma)} E_\sigma(i)} \tag{20}$$
The same equation holds for $\gamma(\sigma, \epsilon_\sigma)$.
On the other hand, $M^{[1]}$ keeps a counter $C^{[1]}(\sigma)$ storing the number of nodes labeled with σ in the sample. They are used to assign a probability to every symbol
$$\alpha(1,\sigma) = \frac{1 + C^{[1]}(\sigma)}{|\Sigma| + \sum_{a \in \Sigma} C^{[1]}(a)} \tag{21}$$

With all these ingredients, the tree compression algorithm executes repeatedly a function, `tcompress` schematically represented in Fig. 4, for every tree τ in the input and the maximal order $k_{\max}$ allowed for the models. The `tcompress`

```
algorithm tcompress(k, τ)            [for k > 1]
   r_encode(r_{k-1}(τ))
   do (∀x subtree of τ in breadth-first order)
      t ← r_k(x)
      if (1 + depth(t) ≥ k − 1) then
         f_encode(k, t)
      endif
   enddo
endalgorithm
```

Fig. 4. Main tree compression algorithm.

algorithm calls a function `f_encode` (plotted in Fig. 5) that encodes the k-forks and $(k-1)$-subtrees[3] and a similar function `r_encode` (plotted in Fig. 7) that encodes the $(k-1)$-root. All these functions use a generic procedure $\texttt{send}(\varphi, \eta, x)$ where $\varphi \in \{\alpha, \beta, \gamma\}$, η is a k- or σ- value and x is a tree or m-value. Both φ and η specify the table to be used (that is, the equation (18–21 and first argument) while x is the parameter that selects the table entry (the second argument in these equations). The function `send` generates the input for the arithmetic encoder, that is, the cumulative range of probabilities assigned to the event by the model.

The corresponding decoding functions are implemented in a similar fashion and can be found in the appendix. We checked this model with a 6.129 Mbyte file consisting of parse trees (with structural and part-of-speech tags) contained in the Penn Tree-bank (Marcus et al., 1993). The compression rate obtained, 13.94, is to be compared with 7.08 obtained using `gzip` with best compression options, 10.92 obtained with `bzip2` with longest block-size or 9.08 using trigram-based arithmetic compression.

5 Multilevel smoothing of tree languages

A standard procedure to avoid null probabilities is the backing-off method that has been extensively studied for string models (see, for instance, Ney et al. (1995)). In this section we introduce a backing-off procedure that has been succesfully used in classification tasks. Different schemes are possible but the results are not very sensitive to the detailed structure (Ney et al., 1995) and our main purpose is to illustrate the difficulties and differences with the string case. Indeed, the most important diference comes from the fact that the number of

[3] this means that the argument t for `f_encode` satisfies $k-1 \leq 1 + \mathrm{depth}(t) \leq k$.

```
function f_encode(k, t)
   q ← r_{k-1}(t)                    [k > 1 is always true]
   if (C^[k](t) > 0) then                    [t found in M^[k]]
      send(α, k, t)
      update(k, t)
   else                   [t not found in M^[k]]
      send(α, k, ε_q)
      inc(C^[k](ε_q))
      if (k > 2) then                    [use M^[k-1] for t = σ(t_1, ..., t_m)]
         do (j = 1, ..., m)
            if (1 + depth(t_j) ≥ k − 2) then
               f_encode(k − 1, t_j)
            endif
         enddo
      else                   [use M^[1] for t = σ(a_1, ..., a_m)]
         do (while m ≥ ME_σ(ε_σ))
            send(γ, σ, ε_σ)
            inc(E_σ(ε_σ))
         enddo
         send(γ, σ, m)
         inc(E_σ(m))
         do (j = 1, ..., m)
            send(α, 1, a_j)
            inc(C^[1](a_j))
         enddo
      endif
      inc(C^[k](t))
   endif
endfunction
```

Fig. 5. Encoding function. The command inc increments the counter by one. Function update is expanded in Fig. 6.

descendents of a node is not bounded: on the one hand a special ground model is needed and, on the other hand, care has to be taken when calculating the normalization factors as the number of unseen events is not known. Following the standard approach a discounting function λ such that $0 < \lambda < 1$ is used for every case that originates a discount term Λ with a normalizing factor F. In the following we will use counters $C^{[k]}$, $D^{[k]}$ and E_σ with the same meaning as in previous section.

The probability of a tree $\tau = \sigma(\tau_1, ..., \tau_m)$ in model $M^{[k]}$ is computed using the definitions (5) and (6) adapted to a k-testable language, so that $p(\tau|M^{[k]}) =$

```
function update(k, t)
    if (k > 2) then                    [t = σ(t_1, ..., t_m)]
       do (j = 1, ..., m)
          if (1 + depth(t_j) ≥ k − 2) then update(k − 1, t_j)
       enddo
    else                  [t = σ(a_1, ..., a_m)]
       inc(E_σ(m))
       do (j = 1, ..., m)
          inc(C^[1](a_j))
       enddo
    endif
    inc(C^[k](t))
endfunction
```

Fig. 6. Function `update`.

```
function r_encode (k, t)
   if (D^[k](t) > 0) then                  [use M^[k]]
      send(β, k, t)
      do(i = 2, ..., k)
         inc(D^[i](r_{i−1}(t)))
         update(i − 1, r_{i−1}(t))
      enddo
   else                  [use M^[k−1]]
      send(β, k, ε_r)
      inc(D^[k](ε_r))
      if (k > 2) then                  [t = σ(t_1, ..., t_m)]
         r_encode (k − 1, r_{k−1}(t))
         if(1 + depth(t) ≥ k − 1) f_encode (k − 1, t)
      else                  [t = σ]
         send(α, 1, σ)
         inc(C^[1](σ))
      endif
      inc(D^[k](t))
   endif
endfunction
```

Fig. 7. Root encoding function.

$\rho^{[k]}(r_{k-1}(\tau))\pi^{[k]}(\tau)$ and

$$\pi^{[k]}(\tau) = \begin{cases} p_m^{[k]}(\sigma, r_{k-1}(\tau_1), ..., r_{k-1}(\tau_m)) \; \prod_{i=1}^{m} \pi^{[k]}(\tau_i) & \\ & \text{if } \tau = \sigma(\tau_1, ..., \tau_m) \wedge m > 0 \\ p_0^{[k]}(\sigma) & \text{if } \tau = \sigma \in \Sigma \end{cases} \tag{22}$$

Sometimes, we will need to predict the number m of descendents of a node labeled σ. For this purpose we define

$$p_m^{[1]}(\sigma) = \begin{cases} \frac{E_\sigma(m)}{C^{[1]}(\sigma)} \left(1 - \lambda_\sigma^{[1]}(m)\right) & \text{if } E_\sigma(m) > 0 \\ \frac{\Lambda_\sigma^{[1]}}{F_\sigma^{[1]}} P_{\mu_\sigma}(m) & \text{if } E_\sigma(m) = 0 \wedge C^{[1]}(\sigma) > 0 \\ P_\mu(m) & \text{otherwise} \end{cases} \tag{23}$$

where $\lambda_\sigma^{[1]}$ are strictly positive discounting functions that generate, for each σ, a global discounting factor

$$\Lambda_\sigma^{[1]} = \frac{1}{C^{[1]}(\sigma)} \sum_m E_\sigma(m) \lambda_\sigma^{[1]}(m) \tag{24}$$

which is normalized with

$$F_\sigma^{[1]} = 1 - \sum_{n: E_\sigma(n) > 0} P_{\mu_\sigma}(n) \tag{25}$$

On the other hand, P_μ (or P_{μ_σ}) is the Poisson distribution with parameter the average number μ of descendents of the nodes in the sample (or the average number μ_σ of descendents of nodes labeled σ), that is

$$\mu_\sigma = \frac{\sum_m m E_\sigma(m)}{\sum_m E_\sigma(m)} \tag{26}$$

Note that $\Lambda_\sigma^{[1]} > 0$ when $C^{[1]}(\sigma) > 0$, as this implies $E_\sigma(m) > 0$ for some m.

The ground model $M^{[1]}$ has as a second ingredient: the probabilities $p^{[1]}(\sigma)$ that a node has label σ:

$$p^{[1]}(\sigma) = \begin{cases} \frac{C^{[1]}(\sigma)}{N} \left(1 - \lambda^{[1]}(\sigma)\right) & \text{if } C^{[1]}(\sigma) > 0 \\ \frac{\Lambda^{[1]}}{F^{[1]}} & \text{otherwise} \end{cases} \tag{27}$$

where $\lambda^{[1]}(\sigma)$ is a strictly positive function, $N = \sum_a C^{[1]}(a)$ is the number of nodes in the sample, the normalizing factor $F^{[1]}$ is the number of different alphabet symbols not found in S and

$$\Lambda^{[1]} = \frac{1}{N} \sum_a C^{[1]}(a) \lambda^{[1]}(a) \tag{28}$$

Note that any non-empty sample (that is, with $N > 0$) has $C^{[1]}(\sigma) > 0$ for some σ and, therefore, $\Lambda^{[1]} > 0$.

We will use the ground model to define a backing-off scheme for the three types of elementary probabilities found in Eq. (22): $p_0(\sigma)$, $p_m(\sigma, t_1, ..., t_m)$ with $m > 0$ and $\rho(t)$. The probabilities of the type $p_0(\sigma)$ satisfy $p_0^{[k]}(\sigma) = 1$ if $k > 2$. If $k = 2$, we define $p_0^{[2]}(\sigma) = p_0^{[1]}(\sigma)$ and they are computed using Eq. (23).

The probabilities of the type $p_m^{[k]}$ with $m > 0$ are defined in a different way if $k = 2$ and if $k > 2$. In the first case and given $t = \sigma(a_1, ..., a_m)$, they are computed as follows

$$p_m^{[2]}(\sigma, a_1, ..., a_m) = \begin{cases} \frac{C^{[2]}(t)}{C^{[1]}(\sigma)} \left(1 - \lambda^{[2]}(t)\right) & \text{if } C^{[2]}(t) > 0 \\ \frac{\Lambda^{[2]}(\sigma)}{F^{[2]}(\sigma)} \, p_m^{[1]}(\sigma) \prod_{i=1}^{m} p^{[1]}(a_i) & \text{otherwise} \end{cases} \tag{29}$$

where

$$\Lambda^{[2]}(\sigma) = \sum_{u: C^{[2]}(u) > 0 \wedge r_1(u) = \sigma} \frac{C^{[2]}(u)}{C^{[1]}(\sigma)} \, \lambda^{[2]}(u) \tag{30}$$

and the normalization factor is

$$F^{[2]}(\sigma) = 1 - p_0^{[2]}(\sigma) - \sum_{m>0: E_\sigma(m)>0} p_m^{[1]}(\sigma) \sum_{\substack{a_1, ..., a_m: \\ C^{[2]}(\sigma(a_1, ..., a_m)) > 0}} \prod_{i=1}^{m} p^{[1]}(a_i) \tag{31}$$

Again, it is easy to realize that the value of any probability of this type is strictly positive.

On the other hand, the probabilities of type $p_m^{[k]}$ with $k > 2$ and $m > 0$, given $t = \sigma(t_1, ..., t_m)$ with $t_i = a_i(s_{i1}, ..., s_{im_i})$, are defined as

$$p_m^{[k]}(\sigma, t_1, ..., t_m) = \begin{cases} \frac{C^{[k]}(t)}{C^{[k-1]}(r_{k-1}(t))} \left(1 - \lambda^{[k]}(t)\right) & \text{if } C^{[k]}(t) > 0 \\ \frac{\Lambda^{[k]}(r_{k-1}(t))}{F^{[k]}(r_{k-1}(t))} \prod_{i=1}^{m} p_{m_i}^{[k-1]}(a_i, s_{i1}, ..., s_{im_i}) & \text{otherwise} \end{cases} \tag{32}$$

where

$$\Lambda^{[k]}(q) = \frac{1}{C^{[k-1]}(t)} \sum_{u: C^{[k]}(u) > 0 \wedge r_{k-1}(u) = q} C^{[k]}(u) \lambda^{[k]}(u) \tag{33}$$

and the normalization factor is

$$F^{[k]}(\sigma(\tau_1, ..., \tau_m)) = 1 - \sum_{\substack{u_1, ..., u_m: \\ C^{[k]}(\sigma(u_1, ..., u_m)) > 0 \wedge r_{k-2}(u_i) = \tau_i}} \prod_{i=1}^{m} p_{m_i}^{[k-1]}(a_i, v_{i1}, ..., v_{im_i}) \tag{34}$$

where $u_i = a_i(v_{i1}, ..., v_{im_i})$.

Finally, $\rho^{[k]}$ probabilities are also needed and a possible discounting scheme in case $k = 2$ is

$$\rho^{[2]}(\sigma) = \begin{cases} \frac{D^{[2]}(\sigma)}{|S|} \left(1 - \theta^{[2]}(\sigma)\right) & \text{if } D^{[2]}(\sigma) > 0 \\ \frac{\Theta^{[2]}}{G^{[2]}} \, p_0^{[1]}(\sigma) & \text{otherwise} \end{cases} \tag{35}$$

where the discounting factor $\Theta^{[2]}$ is

$$\Theta^{[2]} = \frac{1}{|S|} \sum_{a : D^{[2]}(a) > 0} D^{[2]}(a) \theta^{[2]}(a) \tag{36}$$

and its corresponding normalization

$$G^{[2]} = 1 - \sum_{a : D^{[2]}(a) > 0} p^{[1]}(a) \tag{37}$$

In case $k > 2$ we use instead for $t = \sigma(t_1, ..., t_m)$

$$\rho^{[k]}(\sigma(t_1, ..., t_m)) = \begin{cases} \frac{D^{[k]}(t)}{|S|} \; (1 - \theta^{[k]}(t)) & \text{if } D^{[k]}(\sigma(t_1, ..., t_m)) > 0 \\ \frac{\Theta^{[k]}}{G^{[k]}} \; \rho^{[k-1]}(r_{k-2}(t)) \; p_m^{[k-1]}(\sigma, t_1, ..., t_m) & \text{otherwise} \end{cases} \tag{38}$$

where

$$\Theta^{[k]} = \frac{1}{|S|} \sum_{u : D^{[k]}(u) > 0} D^{[k]}(u) \; \theta^{[k]}(u) \tag{39}$$

and the suitable normalizing factor is

$$G^{[k]} = 1 - \sum_{a} \sum_{m} \sum_{\substack{u_1, ..., u_m : \\ D^{[k]}(a(u_1, ..., u_m)) > 0}} \rho^{[k-1]}(r_{k-2}(a(u_1, ..., u_m))) \; p_m^{[k-1]}(a, u_1, ..., u_m) \tag{40}$$

All these equations provide an efficient backing-off scheme that guarantees that no event is assigned a null probability.

6 Conclusion

We have described a probabilistic extension of the k-testable tree languages that can be also regarded as a generalization of k-grams for tree languages. The models of this type can be updated incrementally and may work with medium-sized samples, where some state merging methods (Carrasco et al., 2000) tend to output too simple models. The higher compression rates achieved when processing tree data files show that this class of models can be suitable for practical purposes. We also showed that a discounting scheme can be easily defined for this class of models.

Bibliography

Brown, P. F., Pietra, V. J. D., deSouza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Carrasco, R. C., Forcada, M. L., Ángeles Valdés-Muñoz, M., and Ñeco, R. P. (2000). Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. *Neural Computation*, 12(9):2129–2174.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press.

Chaudhuri, R., Pham, S., and Garcia, O. (1983). Solution of an open problem on probabilistic grammars. *IEEE Transactions on Computers*, 32(8):758–750.

Chung, K. L. (1967). *Markov Chains with Stationary Transition Probabilities*. Springer, Berlin, 2 edition.

Cleary, J. G. and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communicaton*, 32(4):396–402.

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

García, P. (1993). Learning k-testable tree sets from positive data. Technical Report DSIC-ii-1993-46, DSIC, Universidad Politécnica de Valencia.

García, P. and Vidal, E. (1990). Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925.

Jelinek, F. (1998). *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts.

Knuutila, T. (1993). Inference of k-testable tree languages. In Bunke, H., editor, *Advances in Structural and Syntactic Pattern Recognition (Proc. Intl. Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland)*. World Scientific.

Knuutila, T. and Steinby, M. (1994). The inference of tree languages from finite samples: an algebraic approach. *Theoretical Computer Science*, 129:337–367.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. (1993). Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19:313–330.

Ney, H., Essen, U., and Kneser, R. (1995). On the estimation of small probabilities by leaving-one-out. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(12):1202–1212.

Nivat, M. and Podelski, A. (1997). Minimal ascending and descending tree automata. *SIAM Journal on Computing*, 26(1):39–58.

Rozenberg, G. and Salomaa, A., editors (1997). *Handbook of Formal Languages*. Springer.

Rubin, F. (1976). Experiments in text file compression. *Communications of the ACM*, 19(11):617–623.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60.
Sánchez, J. and Benedí, J. (1997). Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055.
Stolcke, A. (1995). An efficient context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.
Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kauffman Publishing, San Francisco, 2nd edition.
Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540.
Yokomori, T. (1995). On polynomial-time learnability in the limit of strictly deterministic automata. *Machine Learning*, 19(2):153–179.

A Decoding functions

The corresponding decoding functions can be found in Fig. 9 and 8. These functions call procedures get that play the inverse role of function `send` in compression.

```
function r_decode (k)
   t ← get(β, k)
   if (t! = ε_r) then
      do(i = 2, ..., k)
         inc(D^[i](r_{i-1}(t)))
         update(i − 1, r_{i-1}(t))
      enddo
   else
      inc(D^[k](ε_r))
      if k > 2 then
         q ← r_decode(k − 1)
         t ← f_decode(q)
      else
         σ = get(α, 1)
         inc(C^[1](σ))
         t ← σ
      endif
      inc(D^[k](t))
   endif
   return t
endfunction
```

Fig. 8. Root decoding function.

```
function f_decode (k)
    k ← 2 + depth(q)
    t ← get(α, k, q)
    if (t! = ε_q) then
        update(t)
    else
        inc(C^[k](ε_q))
        if (k > 2) then                         [q = σ(τ_1, ..., τ_m)]
            do (j = 1, ..., m)
                if (1 + depth(τ_j) = k − 1) then
                    t_j = f_decode(τ_j)
                endif
            enddo
            t ← σ(t_1, ..., t_m)
        else                     [q = σ]
            do (while get(γ, σ) = ε_σ)
                inc(E_σ(ε_σ))
            enddo
            m ← get(γ, σ)
            inc(E_σ(m))
            do (j = 1, ..., m)
                a_j ← get(α, 1)
                inc (C^[1](a_j))
            enddo
            t ← σ(a_1, ..., a_m)
        endif
        inc(C^[k](t))
    endif
    return t
endfunction
```

Fig. 9. Decoding function.