ISIT'2013 1569719117

Space-efficient representation of truncated suffix trees, with applications to Markov order estimation*

Luciana Vitale
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
Email: lvitale@fing.edu.uy

Álvaro Martín
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
Email: almartin@fing.edu.uy

Gadiel Seroussi
Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
Email: gseroussi@ieee.org

***Abstract*—Suffix trees (ST) are useful in information-theoretic applications such as model order estimation and lossless source coding, which require access to occurrence counts of patterns of arbitrary length in an input string x. If the length of x, n, is large, the memory required to represent the ST may become a practical performance bottleneck. This can be alleviated, in cases where a nontrivial upper bound is known on the lengths of the patterns of interest, by using a truncated ST (TST). However, conventional TST implementations still require $\Omega(n)$ bits of memory, due to the need to store x. We describe a new TST representation that avoids this limitation by storing all the information necessary to reconstruct the TST edge labels in a string y that is often much shorter than x. We apply TSTs to the implementation of Markov order estimators, where an upper bound k_n on the estimated order is either imposed (for consistency, as in KT-based MDL estimators), or can be derived (as in the BIC estimator). The new representation allows for estimator implementations with sublinear space complexity in some cases of interest. In other cases we show, experimentally, that even when the new representation does not have an asymptotic advantage, it still achieves very significant memory savings in practice.**

I. Introduction

The *suffix tree (ST)* of a string x over a finite alphabet $\mathcal{A}$ is a data structure that makes efficiently available the information on how many times any pattern occurs as a substring of x. Since this is very suitable for collecting statistical data from x, STs are often used as an algorithmic tool in information-theoretic applications, such as source coding [1], [2], [3], [4], prediction [5], classification [6], and model estimation [7]. Efficient algorithms [8], [9], [10] construct the ST of a string of length n in $O(n)$ operations on registers of $O(\log n)$ bits, and require $O(n \log n)$ bits of memory. For large n, these memory requirements and the non-locality of memory accesses can become serious performance bottlenecks in practical implementations, which has motivated the study of more efficient memory representations [11] and alternative constructions [12], [13].

In some applications, there is a known bound on the length of the patterns whose occurrence we are interested in counting, in which case memory requirements can be reduced by constructing a *truncated suffix tree (TST)*. In [3], the well known ST constructions of McCreight [9] and Ukkonen [10] are adapted to build the TST of a string x, and the modified schemes are applied in an implementation of the LZ77 compression algorithm. The empirical memory savings reported by the authors, with respect to the same implementation of LZ77 using STs, are up to 55% in some cases, while slightly reducing the execution time.

In this paper, we study the application of TSTs to Markov order estimation, where, in many cases of interest, there is a natural bound on the order of the candidate models considered and, thus, on the length of the patterns whose statistics need to be collected. A *Minimum Description Length* (MDL) estimator based on the *Krichevskii-Trofimov* (KT) probability assignment [14], for example, is known to be strongly consistent [15] if an upper bound $k_n = o(\log n)$ is imposed on the order of candidate models for an observed sequence of length n, and to be inconsistent if the order estimate is allowed to exceed $C \log n$, for some sufficiently large constant C [16]. The *Bayesian Information Criterion (BIC)* estimator is strongly consistent even if no restriction is imposed on the set of candidate models [16]. In this case, however, it is possible to show a tight uniform upper bound k_n on the estimated Markov order for all individual sequences of length n, which satisfies $|\mathcal{A}|^{k_n} = \Theta(n/\log n)$ [17][18].

To implement a Markov order estimator, we use a TST of depth $k_n{+}1$ to collect the $k{+}1$-st order statistics of the input string x for all orders k up to k_n. These statistics allow us to compute appropriate cost functions for the estimator of interest, and for Markov models of all orders up to k_n from which, in turn, we obtain the order estimate. The number of nodes in the TST is $O\left(|\mathcal{A}|^{k_n}\right)$, and the statistical data associated with each node can be represented using $O(\log n)$ bits. Therefore, if $|\mathcal{A}|^{k_n} = o(n)$, the TST is asymptotically more space-efficient that the ST. Conventional implementations of the TST (e.g., [3]), however, represent substrings of x, which label the edges of the tree, as pairs of indices of symbol locations in x, and store the whole string x to recover the substrings. Thus, this representation has a "floor" of $\Omega(n)$ bits on memory consumption, which dominates the overall space complexity in cases where the storage required by the tree nodes is $o(n)$ (for instance, in the case of the above mentioned MDL estimator with $k_n = o(\log n)$, or, more generally, whenever $|\mathcal{A}|^{k_n} = o(n/\log n)$).

In this paper, we describe a new representation of the

*Work supported in part by grant I+D CSIC-UdelaR.

TST that avoids this limitation. We present an alternative edge labeling strategy based on a so-called *label string* y, which is derived from x but is often significantly shorter, and which is sequentially constructed together with the TST. We show that all the TST edge labels are in fact substrings of y, avoiding the need to store x, and resulting in significant memory savings, in addition to those obtained by using a TST rather than a ST. Estimation algorithms based on the new TST representation can attain space complexities that are $o(n)$ when $|\mathcal{A}|^{k_n} = o(n/\log n)$. The new representation is presented in Section II.

For the sake of concreteness, we describe, in Section III, a linear time algorithm that uses TSTs to implement a class of *penalized maximum likelihood (PML)* Markov order estimators, with a penalization term $|\mathcal{A}|^k f(n)$ on the maximum log-likelihood for k-th order models.[1] For this class of estimators, of which BIC is a special case, an upper bound k_n on the Markov order estimate satisfies $|\mathcal{A}|^{k_n} = \Theta(n/f(n))$ [17], [18], and we show that the algorithm requires $O\left(\frac{n\log n}{f(n)}\right)$ bits of memory. Thus, when $|\mathcal{A}|^{k_n} = o(n/\log n)$, the algorithm attains sub-linear space complexity. On the other hand, if $|\mathcal{A}|^{k_n} = \Omega(n/\log n)$, as is the case, e.g., when $f(n) = c\log n$ for some constant c (BIC corresponds to the case $c{=}(|\mathcal{A}|-1)/2$), the space complexity is $\Omega(n)$ in the worst case. In these cases, the new representation has no asymptotic advantage over the conventional one. Still, we present experimental results on long sequences of "real world" data, showing that a BIC estimator based on the new representation maintains an advantage in memory consumption ranging from 3:1 to 200:1 compared to an estimator based on the conventional TST representation.

Alternatively, the statistical data can be collected in a *full balanced $|\mathcal{A}|$-ary tree* with $|\mathcal{A}|^{k_n}$ leaves, as in the context tree estimation algorithm of [19]. This scheme requires asymptotically the same amount of memory as a TST with the new representation, provided $|\mathcal{A}|^{k_n} = O(n)$. Again, however, our experiments with the BIC estimator show a practical advantage in memory consumption for the new TST implementation over a full balanced $|\mathcal{A}|$-ary tree, ranging from 2:1 to 80:1.

II. Truncated suffix trees

Let $\mathcal{A}$ be a finite alphabet. We denote by $\mathcal{A}^\star$ the set of finite strings over $\mathcal{A}$, and we let $\mathcal{A}^+ = \mathcal{A}^\star \setminus \{\lambda\}$, where λ denotes the empty string. We use $|\cdot|$ to denote string length or set cardinality. For $u, v \in \mathcal{A}^\star$, uv denotes concatenation, $u \preceq uv$ the *prefix* relation, and $uv\backslash u$ the *suffix* v of uv. Any of $u, v, w \in \mathcal{A}^\star$ are *factors* of $z = uvw$; we write $v \in z$ and we say that v *occurs* in position $|u|$ + 1 of z.

A (compact) *$\mathcal{A}^+$-tree* [20] is a rooted tree, with edges labeled with strings from $\mathcal{A}^+$, such that every internal node (except possibly the root) has at least two children, and the labels of every two edges departing from the same node start with different symbols from $\mathcal{A}$ (thus, every node has at most $|\mathcal{A}|$ children). If u is the parent of v, the edge joining u with v is denoted $u \to v$. We identify a node u with the string obtained by concatenating the edge labels in the path from the root to u. We denote by $|T|$ the number of nodes of an $\mathcal{A}^+$-tree T, and we write $u \in T$ if and only if u is a node of T. A string v *is a word* of T if $v \preceq u$ for some node $u \in T$.

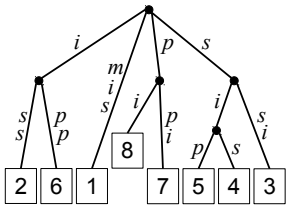

Fig. 1. $TST_k(x)$ for $k = 3$ and $x = mississippi$. The leaf labels are discussed in Section II-B.

A (compact) *suffix tree* [9] of a string x, denoted $ST(x)$, is an $\mathcal{A}^+$-tree such that all the factors of x, but no other strings, are words of $ST(x)$. For a positive integer k, the *truncated suffix tree (TST)* of depth k of x [3], denoted $TST_k(x)$, is the truncation to depth k of $ST(x)$. In other words, $TST_k(x)$ is an $\mathcal{A}^+$-tree such that all the factors of length at most k of x, and no other strings, are words of $TST_k(x)$. We omit the argument x of $ST(x)$ or $TST_k(x)$ when clear from the context. Figure 1 shows an example of a TST for $k = 3$ and $x = mississippi$. We will follow this example in the sequel.

In the rest of this section, we consider a fixed string x of length n over $\mathcal{A}$ and a fixed positive integer k.

A. The label string

Let $x = x_1 x_2 \ldots x_n$. For integers i, j, $i \le j$, we denote by x_i^j the factor $x_{\max\{i,1\}} \ldots x_{\min\{j,n\}}$ of x, and we let $x_i^j = \lambda$ if $i > j$. We write simply x^j to denote x_1^j. For $-(k{-}2) \le i \le n$, we define the *k-factor* in position i of x as $f_i^{(k)} = x_i^{i+k-1}$, i.e., $f_i^{(k)}$ denotes a factor of x that, except in border situations, is of length k. The *head* of $f_i^{(k)}$, denoted $h_i^{(k)}$, is the longest prefix of $f_i^{(k)}$ such that $h_i^{(k)} \preceq f_j^{(k)}$ for some j, $1 \le j < i$, with the convention $h_i^{(k)} = \lambda$ if $i \le 1$. Clearly, if $f_i^{(k)}$ also occurs in an earlier position j, $1 \le j < i$, then $h_i^{(k)} = f_i^{(k)}$. If $h_i^{(k)} \ne f_i^{(k)}$, we say that i is a *first occurrence point* or, for short, a *fop*. In the example, the k-factors $f_i^{(k)}$, for $1 \le i \le 5$, are mis, iss, ssi, sis, iss (resp.), and the corresponding heads are λ, λ, λ, s, iss (resp.). Thus, 1,2,3,4 are fops but 5 is not.

The *label string* of x is a string, $y = y(x^n)$, recursively defined as $y(x^0) = \lambda$ and, for all i, $1 \le i \le n$,

$$y(x^i) = \begin{cases} y(x^{i-1})x_i, & \text{if } i-k+1 \text{ is a fop}, \\ y(x^{i-1}), & \text{otherwise}. \end{cases} \quad (1)$$

In words, we construct y by adding the last symbol of each new k-factor that we find as we scan x up to position $n{-}k{+}1$. In the example, we have $y = missisppi$. We will show that all the edge labels of TST_k are factors of y. Thus, edge labels of TST_k can be represented by pairs of indices of locations in y, rather than x, which, as we will also show, can result

[1] As will be discussed in the full paper, the algorithm can readily be adapted to other estimators of interest, such as the mentioned KT version of MDL, or context tree estimators based on either PML or KT.

in significant memory savings. We notice, however, that in general $TST_k(y) \neq TST_k(x)$; in our example, isp is a factor of y but not of x.

B. Edge labeling

Let N_i denote the number of fops up to position $i \leq n$, i.e.,

$$N_i = \left| \{\, j \,:\, 1 \leq j \leq i,\ f_j^{(k)} \notin x^{j+k-2} \} \right| . \tag{2}$$

In our example, the values of N_i, for $1 \leq i \leq 11$, are 1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 8 (resp.). Figure 1 shows the value of N_i for the first occurrence position i of u, for each leaf u.

The following theorem gives a simple rule to find an edge label of TST_k within y. The proof is given in the Appendix.

Theorem 1. *Let $u \to v$ be an edge of TST_k and let i be the smallest positive integer such that $v \preceq f_i^{(k)}$. The label of $u \to v$ occurs in position $N_i + |u|$ of y.*

C. Implementation details

TST construction algorithms [3] add the k-factors of x to TST_k sequentially, in the same order as they occur in x. Consequently, these algorithms can easily be adapted to construct y sequentially, "on the fly," at the same time as TST_k. Indeed, the insertion of a new leaf during the construction of TST_k, corresponding to some factor $f_{i-k+1}^{(k)}$, $k \leq i \leq n+k-1$, determines, by (1), the concatenation of x_i to the part of y constructed so far. During the construction, the input string x is read sequentially, and need not be stored.

In addition, since each fop gives rise to the creation of a new leaf, if $u \to v$ is an edge where v is the j-th leaf created, then the value of N_i in Theorem 1 is simply j. Thus, the label of $u \to v$ occurs in position $j+|u|$ of y. In Figure 1, the numbers labeling the leaves correspond to the leaf creation order. As a consequence, the memory saving techniques described in [11] apply also to TSTs that use y to label edges. Specifically, if a table of leaves is maintained, and leaves are added to the table as they are created, the starting position within y of the label of $u \to v$ is implicitly determined by the position of v in the table and, thus, need not to be explicitly stored. Labeling edges $u \to v$ where v is an internal node is simple as well, using the fact that v must have been created to split a previously existing edge, which had already been labeled.

D. The length of the label string

The following theorem relates the length of y to the size of TST_k. We let L and L_k denote the set of leaves of ST and TST_k, respectively. The proof is deferred to the Appendix.

Theorem 2. *We have $|L_k| \leq |y| < |L_k| + k$. If $x_n \notin x^{n-1}$, then $|y| = |L_k|$.*

Theorem 2 parallels the well known fact that $|L| \leq n$, with equality if $x_n \notin x^{n-1}$. The following corollary shows that the savings in memory for storing y rather than x are of the same order as the difference between $|ST|$ and $|TST_k|$.

Corollary 1. *The label string y satisfies*

$$n - |y| > \frac{1}{2}\Big(|ST| - |TST_k|\Big) - k\,.$$

Proof: Since every internal node of ST has at least two children, we have $|ST| - |TST_k| \leq 2\Big(|L| - |L_k|\Big)$. The claim then follows from the fact that $|L| \leq n$ and the lower bound on $|L_k|$ in Theorem 2. ∎

III. Markov order estimation

We now regard $x = x^n$ as a sample from a Markov process of unknown order. A *penalized maximum likelihood (PML)* Markov order estimate for x is defined as

$$\hat{k} = \arg\min_{k \geq 0} \Big\{ -\log \hat{P}_k(x) + f(n)|\mathcal{A}|^k \Big\}\,, \tag{3}$$

where f is some positive nondecreasing *penalization function*, and $\hat{P}_k(x)$ is the ML probability of x under a k-th order Markov model (with initial state conventions discussed below). Setting $f(n) = \frac{|\mathcal{A}|-1}{2}\log n$, for example, yields the popular *BIC estimator*. The largest integer k satisfying

$$|\mathcal{A}|^k \leq 1 + n\log|\mathcal{A}|/f(n)\,, \tag{4}$$

is an upper bound on $\hat{k}$ [17], denoted k_n, which is asymptotically attainable [18]. Thus, the set of candidate Markov orders in (3) can be limited to the range $0 \leq k \leq k_n$. In general the most computationally challenging step in this search is computing $-\log \hat{P}_k(x)$ for each candidate k, for which we need to collect all statistics of order k of x, for all k, $0 \leq k \leq k_n$. The TST data structure is used for this purpose.

To completely specify a Markov model for each candidate order k, we fix an *initial string* z, of length k_n, which, for the purpose of selecting the initial states, we regard as preceding x. For a string s and a symbol $a \in \mathcal{A}$, we let $n_s^{(a)}$ denote the number of indices i such that $x_i = a$ and s is a suffix of zx^{i-1}. We also define $n_s = \sum_{a\in\mathcal{A}} n_s^{(a)}$. Then, denoting by $\mathcal{A}^k$ the set of strings of length k, we have

$$-\log \hat{P}_k(x) = -\sum_{s\in\mathcal{A}^k, a\in\mathcal{A}} n_s^{(a)} \log \frac{n_s^{(a)}}{n_s}\,, \tag{5}$$

where $0\log 0 = 0$. Defining

$$\tilde{\mathcal{A}}^k = \{\, s \in \mathcal{A}^k \,:\, n_s \neq n_s^{(a)} \text{ for any } a \in \mathcal{A}\}\,,$$

we can rewrite (5) as

$$-\log \hat{P}_k(x) = \sum_{s\in\tilde{\mathcal{A}}^k} n_s \log n_s - \sum_{s\in\tilde{\mathcal{A}}^k, a\in\mathcal{A}} n_s^{(a)} \log n_s^{(a)}\,. \tag{6}$$

Let $T = TST_{k_n+1}(zx\$)$, where \$ is a special symbol that does not belong to $\mathcal{A}$. This symbol will be useful for a uniform treatment of substrings of x in T (in particular, suffixes). For a node v of T, let m_v and m'_v be the number of indices i such that v occurs in position i of $zx\$$ and z, respectively. The following lemma is an immediate consequence of the above definitions.

Lemma 1. *Let $s \in \mathcal{A}^\star$, $|s| \leq k_n$, and $a \in \mathcal{A}$, be such that $sa \in zx\$$. Then, there exists a unique edge $u \to v$ of T satisfying $u \prec sa \preceq v$, and we have $n_s^{(a)} = m_v - m'_v$. We also have $n_{sa} = n_s^{(a)} - \delta$, where $\delta = 1$ if $sa = v$ and there exists an edge $v \to w$ labeled with \$, and $\delta = 0$ otherwise.*

A. An estimation algorithm

Notice that all nonzero terms in the first and second summations in (6) correspond to strings s and sa, respectively, that are factors of $zx\$$. Now, if for some $s \in \mathcal{A}^k$ of the form $s = tb$, $b \in \mathcal{A}$, and a, u, v as in Lemma 1, we have $u \neq s$, so that $u \prec tb \prec sa \preceq v$, then, by Lemma 1, we must have $n_s^{(a)} = n_t^{(b)}$ and also $n_t^{(b)} = n_s$, implying that $s \notin \tilde{\mathcal{A}}^k$. Thus, every $s \in \tilde{\mathcal{A}}^k$ is an internal node of T, for all k, $0 \leq k \leq k_n$. We make use of this fact and Lemma 1 in the following algorithm, **TSTestim**, that determines $\hat{k}$ by computing the left and right summations in (6), for each candidate k, in arrays **A** and **B**, respectively, indexed by k.

1) Set **A**[0] = n, and **A**[k] = 0 for all k, $0 < k \leq k_n$.
   Set **B**[k] = 0 for all k, $0 \leq k \leq k_n$.
2) For each edge $u \to v$ of T,
   a) If $|v| \leq k_n$, add $(m_v - m'_v - \delta)\log(m_v - m'_v - \delta)$ to **A**$\Big[|v|\Big]$, where δ is defined as in Lemma 1.
   b) Add $(m_v - m'_v)\log(m_v - m'_v)$ to **B**$\Big[|u|\Big]$.
3) Set $\hat{k} = \arg\min\Big\{\mathbf{A}[k] - \mathbf{B}[k] + f(n)|\mathcal{A}|^k : 0 \leq k \leq k_n\Big\}$.

We next analyze the time and space complexity of **TSTestim**. For the time complexity, we count computations of logarithms and $f(n)$ as "primitive operations", consistently with similar analyses in the literature (e.g., [7][19]).

Theorem 3. ***TSTestim*** *requires* $O\left(\frac{n\log n}{f(n)}\right)$ *bits of memory and* $O(n)$ *operations on registers of* $O(\log n)$ *bits.*

Proof: Clearly, Step 2 of **TSTestim** is the most time and memory demanding. The set of counts $\{m_v\}_{v\in L}$, where L is the set of leaves of T, can be computed together with T in linear time, as in [3]. Determining the counts $\{m'_v\}_{v\in L}$ only requires keeping track of the leaves added to T during the first stages of the creation of T (see [3]), which can be handled analogously to $\{m_v\}_{v\in L}$. For an internal node u, m_u and m'_u can be recursively determined by summing, respectively, m_v and m'_v over all children v of u. Therefore, all the computations in Step 2 can be performed in a single bottom-up traversal of T. Since every internal node of T has at least two children, we have $|T| < 2|L|$, and, since $|L| \leq |\mathcal{A}|^{k_n+1} + k_n$, we conclude, by (4), that Step 2 requires $O(n)$ operations. Now, each node u of T requires $O(\log n)$ bits of storage (including auxiliary construction data structures such as suffix links [9] and collected statistics) and, by Theorem 2, edge labels can be represented as substrings of a label string that is not longer than $|L| + k_n$. Hence, the space complexity of **TSTestim** is, by (4), $O(n\log n/f(n))$, as claimed. ■

Notice that, by Theorem 3, the space complexity of **TSTestim** is $o(n)$ if $\log n = o(f(n))$, i.e., by (4), if $|\mathcal{A}|^{k_n} = o(n/\log n)$. On the other hand, as mentioned, the space complexity of a conventional TST implementation is $\Omega(n)$. If $f(n) = \Omega(\log n)$ (as in BIC), Theorem 3 yields $O(n)$ space complexity and, therefore, we cannot claim an asymptotic advantage over a conventional representation. However, the new representation never requires more memory space than the conventional one, and, in fact, as shown in the next subsection, the memory savings may still be very significant in practice.

TABLE I
DATA STRUCTURE SIZE

File	n (bits)	k_n	$\lvert y\rvert$ (bits)	$\lvert y\rvert/n$ (%)	$\lvert\mathcal{A}\rvert^{k_n}$	$\lvert T\rvert$
1	842,752	16	3,991	0.47	65,536	7,964
2	1,036,288	16	5,799	0.56	65,536	11,580
3	1,228,800	16	3,830	0.31	65,536	7,643
4	2,593,472	17	11,197	0.43	131,072	22,375
5	12,212,224	19	3,241	0.03	524,288	6,461
6	17,915,904	20	218,077	1.22	1,048,576	436,132
7	21,138,432	20	5,837	0.03	1,048,576	11,652

B. Experimental results

We ran an implementation of **TSTestim**, with $f(n) = \frac{1}{2}\log n$ (i.e., BIC), on a set of "real world" black and white binary halftone images. Table I shows the length of the label string y for the TST T of depth $k_n + 1$ built in each case, where n denotes the the bit size of the image. The length of y is 1.22% or less (often, significantly so) of the length of the input string in all cases. Notice, also from Table I, that the number of nodes in T is generally much smaller than the number of different patterns of length k_n, namely, $|\mathcal{A}|^{k_n}$. Consequently, as mentioned in Section I, computing T may be more memory efficient than building a full balanced tree of depth k_n, even though the worst case memory requirements of the two schemes are asymptotically the same.

TABLE II
MEMORY REQUIREMENTS

File	T_{full} (Bytes)	T_{old} (Bytes)	T_{new} (Bytes)	$T_{\text{new}}/T_{\text{full}}$ (%)	$T_{\text{new}}/T_{\text{old}}$ (%)
1	524,288	901,623	73,043	13.93	8.10
2	524,288	1,121,455	106,637	20.34	9.51
3	524,288	223,536	70,415	13.43	31.50
4	1,048,576	2,758,373	205,696	19.62	7.46
5	4,194,304	12,262,793	56,454	1.35	0.46
6	8,388,608	20,969,269	4,170,084	49.71	19.89
7	8,388,608	21,229,765	101,666	1.21	0.48

The overall memory savings obtained by using y instead of x depend on the specific memory representation of the tree and the associated statistics. Table II compares the memory required to store the TST and the set of counts $\{m_v\}_{v\in L}$,[2] where the nodes of T are represented as in [11] and each count m_v is represented using 32 bits, in both the conventional and the new TST representations, referred to as T_{old} and T_{new}, respectively. Notice that, using T_{new}, **TSTestim** requires, for all of the tested images, less than $1/3$ of the storage required by the same algorithm using T_{old}, with the savings being significantly larger in many cases.

[2]The set of counts $\{m'_v\}_{v\in L}$ can be determined by keeping track of the first k_n leaves created in T, which requires negligible memory space.

To fairly compare the memory requirements of **TSTestim** against an implementation that constructs a full balanced tree of depth k_n (denoted $T_{\rm full}$) as in [19], we must consider the fact that a TST requires auxiliary data structures that are not needed to represent a full balanced tree. If we just store the counts $n_s^{(a)}$ for all $s\in\mathcal{A}^{k_n}$, $a\in\mathcal{A}$, in an array of $|\mathcal{A}|^{k_n+1}$ integers (again with 32 bit representations), the memory requirements are those shown for $T_{\rm full}$ in Table II. The new TST implementation requires less than half the space required by an implementation based on $T_{\rm full}$ for all the tested images, and, again, the savings are much greater in many cases.

Appendix A
Proofs of Theorems 1 and 2

We make use of the following two lemmas.

Lemma 2 ([9], [3])**.** *For all i, $1 < i \le n$, we have $h_{i-1}^{(k)} \preceq x_{i-1}h_i^{(k)}$.*

Lemma 3. *If $i-k+1$ is a fop, $k \le i \le n+k-1$, then $f_{i-k+1}^{(k)}\backslash h_{i-k+1}^{(k)}$ is a suffix of $y(x^i)$, and we have $|y(x^i)| = N_{i-k+1} + |f_{i-k+1}^{(k)}| - 1$.*

Proof: We first notice that for $i=k$, the index $i-k+1 = 1$ is a fop that satisfies the claim. Indeed, we have $f_1^{(k)}\backslash h_1^{(k)} = f_1^{(k)} = y(x^k)$ and $N_1 = 1$.

Assume now that $i > k$, let j be the largest index smaller than i such that $j-k+1$ is a fop, and assume, by induction, that the lemma holds for j in the role of i. We first show that $|y(x^i)| = N_{i-k+1} + |f_{i-k+1}^{(k)}| - 1$. If $i \le n$, then $|f_{i-k+1}^{(k)}| = |f_{j-k+1}^{(k)}| = k$, and $|y(x^i)| - |y(x^j)| = N_{i-k+1} - N_{j-k+1} = 1$, which, by the induction assumption, implies that $|y(x^i)| = N_{i-k+1} + |f_{i-k+1}^{(k)}| - 1$. If otherwise $i > n$, $f_{i-k+1}^{(k)}$ is a suffix of $f_{i-k}^{(k)}$, implying, since $i-k+1$ is a fop, that $i-k$ is also a fop and, thus, $j = i-1$. We then have $N_{i-k+1} - N_{j-k+1} = |f_{j-k+1}^{(k)}| - |f_{i-k+1}^{(k)}| = 1$ and $|y(x^i)| = |y(x^j)|$, yielding, again, $|y(x^i)| = N_{i-k+1} + |f_{i-k+1}^{(k)}| - 1$ as claimed.

Now, by Lemma 2, $f_{i-k+1}^{(k)}\backslash h_{i-k+1}^{(k)}$ is a suffix of $f_{i-k}^{(k+1)}\backslash h_{i-k}^{(k)}$, which we claim to be a suffix of $y(x^i)$. Let $z = x_i$ if $i \le n$, and $z = \lambda$ otherwise. We then have

$$f_{i-k}^{(k+1)}\backslash h_{i-k}^{(k)} = \left(f_{i-k}^{(k)}\backslash h_{i-k}^{(k)}\right)z\,, \tag{7}$$

and, since $i-k+1$ is a fop, by (1), recalling that $x^i = x^n$ if $i > n$, we also have

$$y(x^i) = y(x^{i-1})z\,. \tag{8}$$

If $i-k$ is a fop, by the induction assumption, $f_{i-k}^{(k)}\backslash h_{i-k}^{(k)}$ is a suffix of $y(x^{i-1})$, implying, by (7) and (8), that $f_{i-k}^{(k+1)}\backslash h_{i-k}^{(k)}$ is a suffix of $y(x^i)$ as claimed. Otherwise, if $i-k$ is not a fop, then $f_{i-k}^{(k)}$ equals $h_{i-k}^{(k)}$, and the claim follows again by (7) and (8). ∎

Proof of Theorem 1: Clearly, by the definition of i, we have $h_i^{(k)} \preceq v \preceq f_i^{(k)}$. Moreover, since $h_i^{(k)} \preceq f_j^{(k)}$ for some $j < i$, we must have $h_i^{(k)} \preceq u$, for otherwise there would exist a node splitting the edge $u \to v$. Hence, $f_i^{(k)}\backslash u$ is a suffix of $f_i^{(k)}\backslash h_i^{(k)}$, which, by Lemma 3, is a suffix of $y^{N_i+|f_i^{(k)}|-1}$. As a consequence, the edge label $v\backslash u$, which is a prefix of $f_i^{(k)}\backslash u$, is a factor of y occurring in position

$$N_i + |f_i^{(k)}| - 1 - (|f_i^{(k)}| - |u|) + 1 = N_i + |u|\,.$$

∎

Proof of Theorem 2: If $n < k$, then $y = x$, $TST_k = ST$, and the claim is thus obvious. We assume, therefore, that $n \ge k$. Let i be the largest integer, $k \le i \le n$, such that $i-k+1$ is a fop. Then, we have $y = y(x^i)$ and, by Lemma 3, the length of the label string is $|y| = N_{i-k+1} + |f_{i-k+1}^{(k)}| - 1$. By definition of N_i, we have $|L_k| = N_{i-k+1} + |\tilde{L}_k|$, where $\tilde{L}_k$ denotes the set of leaves u of TST_k such that $|u| < k$. We then have

$$|y| = |L_k| - |\tilde{L}_k| + |f_{i-k+1}^{(k)}| - 1\,,$$

which, since $0 \le |\tilde{L}_k| < k$, and $|\tilde{L}_k| = k-1$ if $x_n \notin x^{n-1}$, completes the proof. ∎

References

[1] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," Digital SRC Research Report, Tech. Rep., 1994.

[2] M. Effros, "PPM performance with BWT complexity: A fast and effective data compression algorithm," *Proc. IEEE*, vol. 88, pp. 1703–1712, 2000.

[3] J. C. Na, A. Apostolico, C. S. Iliopoulos, and K. Park, "Truncated suffix trees and their application to data compression," *Theoretical Computer Science*, vol. 304, no. 1–3, pp. 87–101, 2003.

[4] Á. Martín, G. Seroussi, and M. J. Weinberger, "Linear time universal coding and time reversal of tree sources via FSM closure," *IEEE Trans. Inform. Theory*, vol. 50, pp. 1442–1468, 2004.

[5] P. Jacquet, W. Szpankowski, and I. Apostol, "A universal predictor based on pattern matching," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1462–1472, 2002.

[6] J. Ziv, "On finite memory universal data compression and classification of individual sequences," *IEEE Trans. Inform. Theory*, vol. 54, pp. 1626–1636, 2008.

[7] A. Garivier, "Consistency of the unlimited BIC context tree estimator," *IEEE Trans. Inform. Theory*, vol. 52, pp. 4630–4635, 2006.

[8] P. Weiner, "Linear pattern matching algorithms," in *Proc. 14th Symp. Switching and Automata Theory*, Washington, DC, 1973, pp. 1–11.

[9] E. M. McCreight, "A space-economical suffix tree construction algorithm," *J. ACM*, vol. 23, pp. 262–272, 1976.

[10] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, pp. 249–260, 1995.

[11] S. Kurtz, "Reducing the space requirement of suffix trees," *Software Pract. Exper.*, vol. 29, pp. 1149–1171, 1999.

[12] R. Giegerich, S. Kurtz, and J. Stoye, "Efficient Implementation of Lazy Suffix Trees," *Software Pract. Exper.*, vol. 33, pp. 1035–1049, 2003.

[13] S. Tata, R. A. Hankins, and J. M. Patel, "Practical suffix tree construction," in *Proc. 13th Int. Conf. VLDB*, 2004, pp. 36–47.

[14] R. E. Krichevskii and V. K. Trofimov, "The performance of universal encoding," *IEEE Trans. Inform. Theory*, vol. 27, pp. 199–207, 1981.

[15] I. Csiszár, "Large-scale typicality of Markov sample paths and consistency of MDL order estimators." *IEEE Trans. Inform. Theory*, vol. 48, pp. 1616–1628, 2002.

[16] I. Csiszár and P. C. Shields, "The consistency of the BIC Markov order estimator." *Annals of Statistics*, vol. 28, pp. 1601–1619, 2000.

[17] Á. Martín, N. Merhav, G. Seroussi, and M. J. Weinberger, "Twice-universal simulation of Markov sources and individual sequences," *IEEE Trans. Inform. Theory*, vol. 56, pp. 4245–4255, 2010.

[18] L. Vitale, Á. Martín, and G. Seroussi, "Bounds on estimated Markov orders of individual sequences," in *Proc. ISIT*, 2012, pp. 1102–1106.

[19] I. Csiszár and Z. Talata, "Context tree estimation for not necessarily finite memory processes, via BIC and MDL," *IEEE Trans. Inform. Theory*, vol. 52, pp. 1007–1016, 2006.

[20] R. Giegerich and S. Kurtz, "From Ukkonen to McCreight and Weiner: A unifying view to linear-time suffix tree construction," *Algorithmica*, vol. 19, pp. 331–353, 1997.