

Approximate validity of XML Streaming Data

Huang Cheng ^{#1}, Li Jun ^{#2}, Michel de Rougemont ^{*3}

[#]Univ. Paris-Sud & Huazhong Univ.

LRI, 91405 Orsay, France & 430074 Wuhan, China

¹cheng.huang@u-psud.fr

²jun.li@u-psud.fr

^{*}Univ. Paris II

Paris, France

³mdr@lri.fr

Abstract— We present a SAX implementation of the statistical embedding associated with XML data, introduced in [1], [2], which allows to efficiently decide ε -validity to any DTD or Schema, for the Edit Distance with Moves. It associates a generalized k -gram to unranked labelled trees (with $k = \frac{1}{\varepsilon}$) from which any regular property can be approximately decided. We show how to exactly compute the k -gram with a SAX implementation using a memory of size d , the depth of the tree, and an approximate k -gram with queues of size $M = 2k$ and a global memory of size 2^k in the worst-case. Experiments on large XML files from the XML benchmark project confirm the error analysis for various values of M .

I. INTRODUCTION

An important problem for Web mining is to decide if semistructured documents are close to some DTDs, as Web documents are noisy and may need to be transformed before they are analyzed. Approximate validity has been studied in [3] in the context of Property testing for the Edit distance with Moves and further extended in [1]. It decides if a file is close or far from a DTD, by just sampling the file, i.e. in time independent of the length of the file. The algorithms construct sketches which are statistical representations which generalize a k -gram on words.

Streaming algorithms for search problems, as considered in [4] and [5], read the data online and construct an answer using sketches which only require a finite memory. For exact decision problems, these algorithms read the data online and decide at any stage if the input satisfies the property. Exact validity of streaming XML documents has been studied in [6] for some classes of DTDs. For approximate decision problems, these algorithms read the data online and decide at any stage if the input is close or far from some property. A tester may directly yield a streaming algorithm, but in some cases it may require additional work before we obtain a streaming algorithm which uses constant space. If we want to decide if a word is close to a regular expression, the tester used in [1] builds a k -gram, i.e. a density vector which provides the number of subwords of size k , which can be built online and we obtain a streaming algorithm. In the case of trees, we can build the generalized k -gram from a DOM representation of a tree t , but it requires $O(n)$ space. We need a SAX representation which uses constant space and avoids the DOM representation.

In this paper we show how to build an approximate generalized k -gram in constant space, and an exact k -gram in space proportional to the depth d of the tree. The approximate sketch is enough to approximate the validity to any DTD or Schema.

In section 2, we review the notion of approximate validity provided by property testers and the *ustat* representations for words and trees. In section 3, we describe our SAX implementation and prove its correctness. In section 4, we give examples for XML files of different sizes, up to 10^6 generated by the XML benchmark project [7], to show the scalability and the error analysis.

II. PRELIMINARIES

We first describe *approximate validity*, as considered in Property testing. We then define the sketches associated to words and to trees, which allow to approximatedly decide any regular property. We then recall how a union of polytopes of statistical vectors can be associated to any DTD to define a membership tester.

A. Approximate decision problems

This notion of *Property Testing* has been initially defined in [8] and studied for graph properties in [9]. It has been successfully extended to various classes of finite structures, such as words where regular languages are proved testable [10] for the Hamming distance, and trees where regular tree languages are proved testable [3] for the Edit Distance with moves. A tester decides if an input structure satisfies a property or is far from this property, but also requires that we look at a constant fraction of the input, independent of the global size of the input.

We say that two unary structures $U_n, V_m \in \mathbf{K}$ such as words (respectively trees), whose domains are respectively of size n and m , are ε -close if their distance $\text{dist}(U_n, V_m)$ is less than $\varepsilon \times \max(n, m)$. They are ε -far if they are not ε -close. All distances are relative. The distance of a structure U to a class \mathbf{K} is $\text{dist}(U, \mathbf{K}) = \text{Min}_{V \in \mathbf{K}} \{\text{dist}(U, V)\}$. The *Edit distance with moves* between two strings w and w' , written $\text{dist}(w, w')$, is the minimal number of elementary operations on w to obtain w' , divided by $\max\{|w|, |w'|\}$. An *elementary operation* on a word w is either an *insertion*, a *deletion* of a letter, a *modification* of a letter or of an attribute value, or

a *move* of a subword of w into another position. The *Edit distance with moves* between two ordered unranked trees t and t' , written $\text{dist}(t, t')$, is the minimal number of elementary operations on t to obtain t' , divided by $\max\{|t|, |t'|\}$. An *elementary operation* on a tree t is either an *insertion*, a *deletion* of a node or of an edge, a *modification* of a letter (tag) or of an attribute value, or a *move* of an entire subtree of t into another position. When an XML file is given by its DOM representation, these operations take unit costs. Two schemas S_1 and S_2 on the same alphabet are ε -close if all but finitely many words or trees of S_1 (resp. S_2) are ε -close to S_2 (resp. S_1).

Definition 1: Let $\varepsilon \geq 0$ be a real. An ε -tester for a class $\mathbf{K}_0 \subseteq \mathbf{K}$ is a randomized algorithm A such that:

- (1) If $U_n \in \mathbf{K}_0$, A always accepts;
- (2) If U_n is ε -far from \mathbf{K}_0 , then $\Pr[A \text{ rejects}] \geq 2/3$.

The *query complexity* is the number of queries to the structure I of \mathbf{K} : in the case of words, a query asks for the letter in position i . The *time complexity* is the usual time complexity where the complexity of a query is one and the time complexity of an arithmetic operation is also one. A class $\mathbf{K}_0 \subseteq \mathbf{K}$ is *testable* if for every sufficiently small $\varepsilon > 0$, there exists an ε -tester whose time complexity depends only on ε . If we strengthen the first condition (1) to:

- (1') If I is ε' close of \mathbf{K}_0 , A always accepts; we have an $(\varepsilon, \varepsilon')$ *tolerant tester*, for $\varepsilon > \varepsilon'$.

B. Statistical embedding on strings

For a finite alphabet Σ and a given ε , let $k = \frac{1}{\varepsilon}$. Let the $\text{dist}(w, w')$ be the *Edit distance with moves* and the embedding of a word in a vector of k -statistics, describing the number of occurrences of all subwords of length k in w . Let w be a word of length n , let $\#u$ be the number of occurrences of u of length k in w and $\text{ustat}(w)[u] \stackrel{\text{def}}{=} \frac{\#u}{n-k+1}$. The vector $\text{ustat}(w)$ is of dimension $|\Sigma|^k$ is also the *probability distribution* that a uniform random subword of size k of w be a specific u , i.e. $\text{ustat}(w)[u] = \Pr_{j=1, \dots, n-k+1}[w[j]w[j+1] \dots w[j+k-1] = u]$. This embedding is a generalized Parikh mapping [11], is also related to [12] where the subwords of length k were called *shingles* and is called a k -gram in statistics. This embedding associates $\{\text{ustat}(w) : w \in r\}$ to a regular expression r , a union of polytopes H in the same space, such that the distance between two vectors (for the L_1 norm) is approximately $\text{dist}(w, w')$ and the distance between a vector and a union of polytopes is approximately $\text{dist}(w, L(r))$. For a simple regular expression such as $(001)^*$, the polytope is a unique summit, the *base vector*, which by definition is $\lim_{n \rightarrow \infty} \text{ustat}((001)^n)$. For a more general regular expression, the polytope is the convex hull of the base vectors, associated with compatible loops. Other embeddings on words [13] and trees [14] depend on the size of the structures.

The $\text{ustat}(w)$ vector can be approximated for the L_1 norm by taking N random samples to define the random variable $\widehat{\text{ustat}}(w)$ which approximates $\text{ustat}(w)$. These techniques yield the simple testers of [1] for Equality(w, w') between two words, and Membership (w, r) between a word w and

a regular expression r . Take $N \in O(\frac{(\ln|\Sigma|)|\Sigma|^{2/\varepsilon}}{\varepsilon^3})$ samples, and let $\widehat{\text{ustat}}(w)$ and $\widehat{\text{ustat}}(w')$ be the $\widehat{\text{ustat}}$ of the samples. For Equality, Reject if $|\widehat{\text{ustat}}(w) - \widehat{\text{ustat}}(w')|_1 \geq \varepsilon$. For Membership, compute the set of polytopes H associated with r in the same space. Reject if the geometrical distance from the point $\widehat{\text{ustat}}(w)$ to the polytope H is greater than ε .

Example. Let $k = 2 = 1/\varepsilon$, $w = 000111$, $S = (001)^*. (01)^*. 1^*. (011)^*$ with $\Sigma_S = \{0, 1\}$. For a lexicographic enumeration of the length 2 binary words, $\text{ustat}(w) = (2/5, 1/5, 0, 2/5)$. Let $s_0 = (1/3, 1/3, 1/3, 0)$ the base vector of the regular expression $(001)^*$, and similarly $s_1 = (0, 1/2, 1/2, 0)$ for $(01)^*$, $s_2 = (0, 0, 0, 1)$ for 1^* and $s_3 = (0, 1/3, 1/3, 1/3)$ for $(011)^*$. The polytope H associated with S is $\text{Convex-Hull}(s_0, s_1, s_2, s_3)$ and describes the set of $\text{ustat}(w)$ when $w \in S$ as in figure 1. The word w is at distance $1/6$ to S as it requires the removal of the first 0 to yield the corrected word $001.11 \in S$. For a given w , the geometrical distance between $\text{ustat}(w)$ and H is approximately $\text{dist}(w, S)$ and $\text{dist}(w, w')$ is approximately the L_1 norm between the ustat vectors. For $w_1 = 0101111 = (01)^2.1^3 \in S$, $\text{ustat}(w_1)$ is precisely between the base vectors s_1 and s_2 .

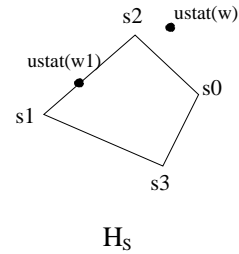


Fig. 1. The statistical space for S of dimension 4 and the polytope H_S for $k = 2$.

C. Statistical embedding on trees

The *size* of a tree is the number n of its nodes. The *degree* of a node is the number of its successors. Let k be an integer and $\varepsilon = 1/k$, as before. Every unranked tree t can be encoded as a binary tree but many encodings are possible. Consider the Rabin encoding $e(t)$ where each node v of the unranked tree is a node v in the binary encoding. Its left successor of in the binary tree is its first successor in the unranked tree, and its right successor in the binary tree is its first sibling in the unranked tree (See Figure 2).

It is a classical observation that a language L of unranked trees is regular iff the language L' of its encoding is regular. An *extended 2-ranked tree* is a binary tree with a left successor, a right successor or both as in figure 2. In the Rabin encoding we obtain extended 2-ranked trees, as some nodes may only have a left or a right successor.

The distance between two unranked trees is of the same order as the distance on their encodings as extended 2-ranked trees. Moreover one can sample any k -subtree of $e(t)$, i.e. subtree of depth k , since one can compute the k -subtree of $e(t)$ from any node v by using 2^k queries asking for the left successor or the sibling of a node v .

The tree embedding introduced in [1] maps the original tree t to a word with labels w_t on a larger alphabet, and then

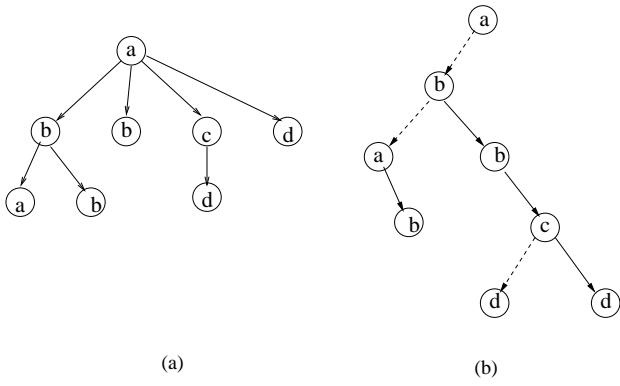


Fig. 2. Encoding of an unranked tree t in (a), as an extended 2-ranked tree $e(t) = t'$ in (b).

uses $\text{ustat}(w_t)$ for the analysis. We propose a more direct approach which avoids some of the encodings and captures a natural statistics. It generalizes $\text{ustat}(w)$ to a matrix $\text{ustat}(t)$ with 2^{k-1} columns. Consider the Rabin encoding of the tree t as in figure 2: paths of length k can be paths on the right successor, i.e. horizontal paths in t , paths on the left successor, i.e. vertical paths in t , or zigzags. There are 2^{k-1} types of paths, and for each type we keep the classical ustat vector. For paths of length k , we associate their type as a boolean vector of length $k-1$. We use 0 for the left branch and 1 for the right branch. For example, the path $abab$ of length $k=4$ in figure 2 is of type 001. We may also represent this path as $abab\bar{}$ on a larger alphabet $\Sigma' = \{a, b, \dots, \bar{a}, \bar{b}, \dots\}$. Let $\text{ustat}_k(t)$ be the matrix with 2^{k-1} columns and $|\Sigma|^k$ lines. In column 1, we have the densities of paths of type 0..0, i.e. vertical paths in the original unranked tree. The last column describes paths of type 1..1, i.e. horizontal paths in the original unranked tree. This *generalized k-gram* also gives the relative density of the paths types, the *global statistics* gstat , a density vector of size 2^{k-1} . The *local statistics* for a given type is the corresponding column vector of the matrix. As the matrix is sparse we only enumerate some of the entries with their non zero probabilities.

Example. Consider the XML file associated with the figure 3, following the DTD S . For a better presentation, many null entries are not represented in the matrices and we abbreviate “author” by a , “bd” and “bib” by b , “work” by w . The Source Schema S is:

```
<!ELEMENT db (work*)>
<!ELEMENT work (author*)>
<!ATTLIST work title CDATA year CDATA>
<!ELEMENT author (EMPTY)>
<!ATTLIST author name CDATA #REQUIRED>
```

We give below the two matrices for $k=2$ and $k=3$ for the tree t of figure 3.

$$\text{ustat}_2(t) = \frac{1}{5} \begin{pmatrix} aa : 0 & a\bar{a} : 1 \\ bw : 1 & b\bar{w} : 0 \\ wa : 2 & w\bar{a} : 0 \\ ww : 0 & w\bar{w} : 1 \end{pmatrix}$$

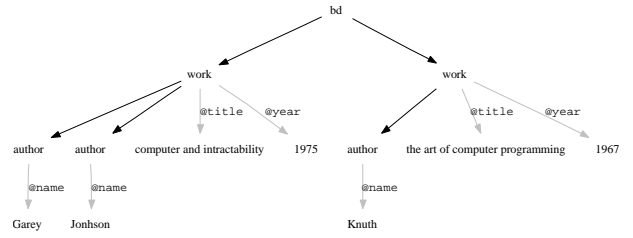


Fig. 3. The tree t associated with some file $F.xml$.

$$\text{ustat}_3(t) = \frac{1}{4} \begin{pmatrix} bwa : 1 & bw\bar{a} : 0 & b\bar{w}a : 0 & b\bar{w}\bar{a} : 0 \\ bbw : 0 & bw\bar{w} : 1 & b\bar{w}w : 0 & b\bar{w}\bar{w} : 0 \\ waa : 0 & wa\bar{a} : 1 & w\bar{a}a : 0 & w\bar{a}\bar{a} : 0 \\ wwa : 0 & ww\bar{a} : 0 & w\bar{w}a : 1 & w\bar{w}\bar{a} : 0 \end{pmatrix}$$

For $k=2$, then $\text{gstat}_2(t) = \frac{1}{5} \begin{pmatrix} 0 : 3 \\ 1 : 2 \end{pmatrix}$, where 0 represents the left branch, and 1 represents the right branch. The *local statistic* of the right branch 1 is :

$$\text{lstat}_1 = \frac{1}{2} \begin{pmatrix} a\bar{a} : 1 \\ b\bar{w} : 0 \\ w\bar{a} : 0 \\ w\bar{w} : 1 \end{pmatrix}$$

D. Statistical embedding of DTDs

We associate a set of *base loops* t_i to the DTD S , i.e. irreducible loops t_i , and a set of *base vector* $s_i = \text{ustat}(t_i^*) = \lim_{n \rightarrow \infty} \text{ustat}((t_i)^n)$. The set of $\text{ustat}(t)$ for $t \in S$ is a union of polytope H_i^S which is the Convex-Hull (s_1, \dots, s_l) of the base vectors of compatible base loops, restricted to some additional linear constraints. If $\text{ustat}(t)$ is ε -close to H_i , then it is also close to $\sum_{s_i \in C} \lambda_i \cdot s_i$ where $C \subseteq \{s_1, \dots, s_p\}$ of size at most $d_S + 1$, where d_S is the dimension of the Source vectors, i.e. $2^{k-1} \cdot |\Sigma_S|^k$.

Example. Consider the DTD given by the 4 rules: $\text{root} : a^*b$, $a : c.d$, $c : a.f + g$, $b : e^*$. The simple loops are a^* , e^* and $(a\bar{c}.fd)^*$, where the $c\bar{\cdot}$ indicates that the recursion applies to the left successor of c . The base vectors for $k=2$ are $s_1 = (aa : 1)$, $s_2 = (ee : 1)$ and $s_3 = (cd : 1/4 : af : 1/4, a\bar{c} : 1/4, c\bar{a} : 1/4)$. If t is close to the DTD, then $\text{ustat}(t)$ is close to $\lambda_1 \cdot s_1 + \lambda_2 \cdot s_2 + \lambda_3 \cdot s_3$ for $\sum_i \lambda_i = 1$. The distance from a tree t to the DTD is approximately the geometrical distance from $\text{ustat}(t)$ to H .

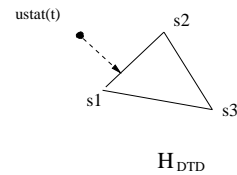


Fig. 4. The statistical space for a DTD and the polytope H_{DTD} for $k=2$.

We obtain a Membership tester between a tree t and a DTD. Take $N \in O(\frac{(\ln|\Sigma|)|\Sigma|^{2/\epsilon}}{\epsilon^3})$ samples, and let $\widehat{\text{ustat}}(t)$ be the $\widehat{\text{ustat}}$ of the samples. We compute the set of polytopes H associated with the DTD. If $\widehat{\text{ustat}}(t) \in H$ we accept and if $\widehat{\text{ustat}}(t)$ is ϵ -far from H we reject. In the next section, we don't sample the tree but read it online and update the $\widehat{\text{ustat}}$ vector at every stage.

III. SAX-IMPLEMENTATION

In the case of streaming data, it is simple to update the $\widehat{\text{ustat}}$ vector in the case of words, by just increasing the counter associated with the new letter read. In the case of trees, it is also simple if we start from a DOM representation. The challenge is to build an approximate $\widehat{\text{ustat}}$ matrix with a bounded memory in an online manner. We first present a SAX implementation which constructs the statistics online, with a memory size proportional to the the depth of the tree. We then show that an approximate statistics $\widehat{\text{sustat}}$ can be obtained with a finite memory only. We describe the data structures and the operations to update them as we read online a new open or close tag p .

A. Data Structures

For a given node with an open tag p , we want to keep the path in the Rabin encoding leading to this node, and then update the statistics. The state of the algorithm consists in 3 vectors: TagPath , TypePath , and Width .

The two vectors TagPath and TypePath represent the path in the Rabin encoding. The former describes the sequence of tags names of the nodes leading to the current one, of length d_R , the depth of the node in the Rabin encoding. The latter describes a binary word of length $d_R - 1$ which specifies left (0) or right (1) branches.

The additional vector Width , whose size is d , the depth of the current node in the unranked tree, maintains the current widths of the unranked tree at the different levels above the current node. The last value w is the current number of siblings of the current node if it has an open tag, i.e. the number of children of his father. This information is kept to indirectly build a Rabin encoding in an online manner. It is used to update the TagPath and TypePath vectors when Pop operations are performed. For example, if $w = 5$ and we return to the closing tag of the father node, the TagPath and TypePath vectors are Popped 5 times. Figure 6 shows the data structures after reading $\langle h \rangle$ in the tree of figure 5. The path a, c, h in the unranked tree, leading to h is of depth 2. Its width is 2 at the first level (b, c) and 2 at the second level (g, h) and $\text{Width} = (2, 2)$.

The global statistics is maintained as a vector gstat of dimension 2^{k-1} . Each value counts the number of subtree of a given type. The Local statistics are 2^{k-1} vectors of dimension $|\Sigma|^k$, as in the case of words.

B. Operations

On each of the vectors TagPath , TypePath , and Width , we apply Pop and Push operations. On the Width vector we

may also increase its last value w , with the instruction $w++$. The Algorithm S with input a file F and a parameter k , with *unbounded memory*, can now be described in a pseudo-code form.

Algorithm $S(F,k)$:

O. For the root, Push the tag on Tagpath ,

1. If the current Input is $\langle t \rangle$ with p the previous tag.

1.1 Push t to Tagpath ,

1.2 If $p = \langle s \rangle$ then Push 1 to Width and Push 0 to Typepath ,

1.3 If $p = \langle /s \rangle$ then $w++$ and Push 1 to Typepath

1.4 Update statistics: increase the gstat counter for the new subtree of type Typepath if the length is greater than k and update the corresponding lstat .

2. If the current Input is $\langle /t \rangle$ with p the previous tag.

2.1 If $p = \langle s \rangle$ then proceed,

2.2 If $p = \langle /s \rangle$ then Pop the last w elements of Tagpath and Typepath , Pop Width .

The memory used is the maximal depth in the Rabin encoding of tree t .

C. Examples

Consider the XML file F associated to the tree in figure 5. Suppose the parser reads $\langle h \rangle$ in F . The previous tag is $\langle /g \rangle$. The algorithm S pushes h into TagPath and pushes 1 (code of the Rabin encoding) into TypePath . The last element w in Width was 1, and becomes 2 after the $w++$ instruction. The state after the operation is shown in figure 6.

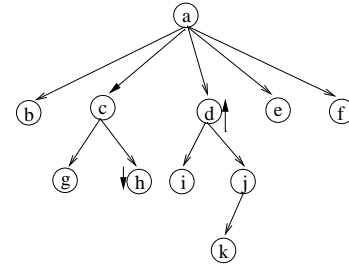


Fig. 5. Tree associated with an XML file.

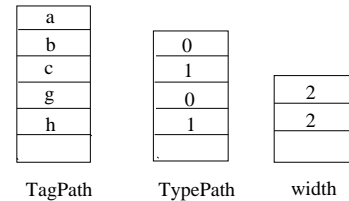
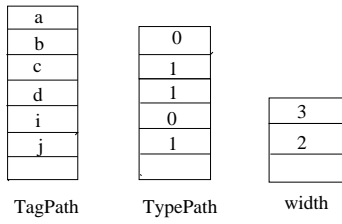
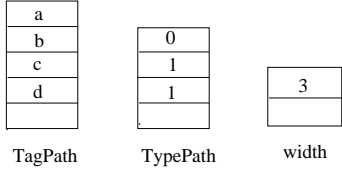


Fig. 6. State after parsing $\langle h \rangle$.

Later in the file we read $\langle /d \rangle$, and the previous tag is $\langle /j \rangle$. The value of w , the last element of Width , is 2. We pop the last 2 elements in TagPath and TypePath . The states are shown in figure 7.

(a) State before parsing $\langle /d \rangle$ (b) State after parsing $\langle /d \rangle$ Fig. 7. State of memory after parsing $\langle /d \rangle$.

D. Analysis for unbounded memory

The fundamental observation of the algorithm S is that $TagPath$ and $TypePath$ describe at any stage, the path in the Rabin encoding which leads to the current node of the tree. We prove this property by induction on the length of the file F and use the variable w maintained in $Width$.

Lemma 1: If a node u at depth d in the unranked tree has m children, then $Width(d) = j$ when we read the open tag of the j -th child of u , for $j \leq m$.

Proof: Let u be a node with children v_1, \dots, v_m . When we read $\langle /v_1 \rangle$, then $Width$ is pushed with 1, and the assertion is verified. Suppose the assertion is true for v_j . When we read $\langle v_{j+1} \rangle$, the previous tag is $\langle /v_j \rangle$ and therefore the S -algorithm realizes $w++$, i.e. increases by 1 the last value of $Width$. The assertion is therefore true for v_{j+1} . We read $\langle /u \rangle$ after we read $\langle /v_m \rangle$, when we Pop $Width$ and therefore the assertion is true for all $j \leq m$. ■

Theorem 1: Let u be the node associated to an open or close tag and π be the path in the Rabin encoding from the root to u . The sequence of tags in $TagPath$ is the sequence of tags of the nodes of π , and the binary word associated with $TypePath$ describes the type of π .

Proof: When we read the first open tag $\langle t \rangle$, from the root, we push t on $TagPath$, and therefore the property is satisfied. Suppose the property is satisfied at stage i , and we want to show it will be satisfied at stage $i+1$. In the first case, we read an open tag $\langle t \rangle$ and the previous tag is $\langle s \rangle$ (resp. $\langle /s \rangle$). We push t on $TagPath$ and 0 (resp 1) on $TypePath$: if the induction hypothesis holds for the predecessor of u , it will hold for u . In the second case, we read a close tag $\langle /t \rangle$ and the previous tag is $\langle t \rangle$ or $\langle /s \rangle$. If the previous tag is $\langle t \rangle$, the node u does not change, and the induction hypothesis

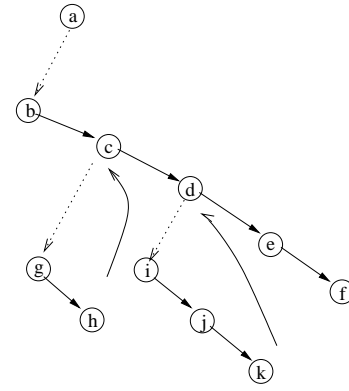


Fig. 8. Sequence of nodes parsed by the Streaming algorithm in the Rabin encoding.

is kept true. If the previous tag is $\langle /s \rangle$ corresponding to a node v , then u is the father of v and u has w children because of lemma 1. We jump back to a previous level in the unranked tree but we jump back w levels in the Rabin encoding as shown in figure 8. We Pop w -times $TagPath$ and $TypePath$ and the new values are precisely the ones needed. ■

E. Analysis for bounded memory

In order to obtain an algorithm which uses a finite memory, we replace the 3 Pushdowns vectors $TagPath$, $TypePath$, and $Width$ by 3 Bounded Queues of size M . If the Pushdown has size M and we Push a value, we lose the first top value of the vector which we replace by the symbol "?" and maintain the maximum size to $M+1$. Let BPush be a Bounded Push on Queues of size M .

The streaming algorithm generalizes the algorithm S of the previous section. An important modification is the instruction 1.4, which manages the overflow symbol "?", as shown in figure 9. We will later find the lost tag when we read the closing tag and may recover the missed values of the statistics matrix. We save many $Tagpath$ vectors with the "?" symbol, of length less than k , stored in a $Tagbuffer$ with at most 2^{k-1} such $Tagpath$ vectors. Similarly $Typebuffer$ will save the corresponding $Typepath$ vectors. There is also a counter which maintains the number of "?" symbols at any given time. For the sake of simplicity, we do not mention it.

The $Tagbuffer$ may allow us to make some statistics update (instruction 2.4) when we have small right branches, i.e. of depth less than $2k$. This will imply that the only errors on the statistics occur when we encounter forks, defined below.

Streaming Algorithm(F, k, M):

0. For the open tag of the root, Push the tag on $Tagpath$,
1. If the current Input is $\langle t \rangle$ with p the previous tag.
 - 1.1 BPush t to $Tagpath$,
 - 1.2 If $p = \langle s \rangle$ then BPush 1 to $Width$ and BPush 0 to $Typepath$,
 - 1.3 If $p = \langle /s \rangle$ then $w++$ and BPush 1 to $Typepath$

right successor of u is a full binary tree and we miss 2^{k-1} words. From the previous lemma the total error is less than $2^{k-1} \cdot \frac{\varepsilon}{2^{k-1}} \cdot n = \varepsilon \cdot n$. ■

Notice that the total memory is used to store the 3 Queues *TagPath*, *TypePath*, and *Width* of size $M = 2k$ and the *Tagbuffer* and *Typebuffer* of size 2^{k-1} .

F. Approximate validity

We can test if a streaming file is close or far from a DTD, by just considering $\text{sustat}(t)$ which approximates $\text{ustat}(t)$.

Streaming Tester: If $\text{sustat}(t)$ is $2\varepsilon \cdot n$ close to the polytope H associated with a DTD accept, else reject.

Lemma 4: If $\text{dist}(\text{sustat}(t), H) \geq 2\varepsilon \cdot n$ then $\text{dist}(t, H) \geq \varepsilon \cdot n$.

Proof: We just observe that $\text{dist}(t, H) = \text{dist}(\text{ustat}(t), H)$. We apply the triangular inequality: $\text{dist}(\text{sustat}(t), H) - |\text{sustat}(t) - \text{ustat}(t)| \leq \text{dist}(t, H) \leq |\text{sustat}(t) - \text{ustat}(t)| + \text{dist}(\text{sustat}(t), H) \leq \varepsilon \cdot n$. As $|\text{sustat}(t) - \text{ustat}(t)| \leq \varepsilon \cdot n$, we conclude that if $\text{dist}(\text{sustat}(t), H) \geq 2\varepsilon \cdot n$ then $\text{dist}(t, H) \geq \varepsilon \cdot n$. ■

We can now conclude:

Theorem 3: The Streaming tester satisfies the tester conditions.

Proof: If $t \in \text{DTD}$, then $\text{dist}(\text{ustat}(t), H) \leq \varepsilon \cdot n$ and therefore $\text{dist}(\text{sustat}(t), H) \leq 2\varepsilon \cdot n$ and the tester accepts. If t is ε -far from the DTD, we reject with high probability by lemma 4. ■

In practice, we would consider several DTDs: D_1, \dots, D_m and associate a polytope H to each of them. We can then decide which DTD a given streaming file F is closest to, with a streaming algorithm which uses a finite memory.

IV. IMPLEMENTATION RESULTS

We now describe the results on several files from the XML benchmark project (<http://monetdb.cwi.nl/xml/>), up to $n = 2.10^5$. The program reads a file name, its DTD and a parameter k and output the statistics. Our current implementation (<http://www.up2.fr/xmlstream/>) only implements the algorithm S with queues of size $M = 5k$ with the SAX parser, after uploading the file. It does not yet implement the complex recovery mechanism described for the theoretical analysis. Nevertheless the error remains small. In a future implementation, we will avoid the SAX parser and will not need any DTD. We will directly read the file as a stream.

A. Presentation of the Statistics

We first plot the global statistics, i.e. $\text{gstat}_k(t)$, a vector of size at most 2^{k-1} which describes the density of the subtrees of a given type. This is shown in figure 12 as a histogram in decreasing order: the X-axis describes the types of subtrees and the Y-axis their relative density. As $k = 4$, the highest density (43%) is for the type 000, followed by 18% on type 010. There are only 5 types with a significant density out the possible 8, and we conclude that the tree is rather deep.

If we click on the type in the general statistics, we obtain the *local statistic* lstat which describes the density of each subwords. This is shown in figure 13 for the type 010.

The local statistics gives the relative density of the words, starting with the highest density. The word (SPEECH, SPEAKER, SPEECH, SPEAKER) accounts for 87% of the words of length 4 of this type.

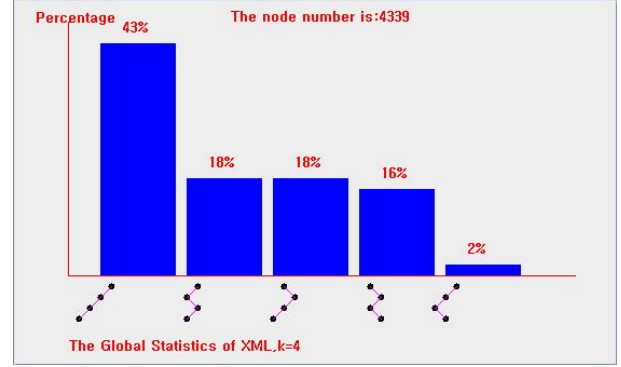


Fig. 12. The global statistics of an XML file for $k = 4$, with only 5 types of trees.

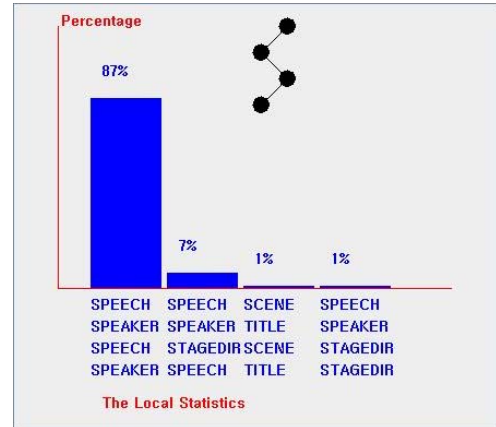


Fig. 13. The local statistics of trees of type 010 with only 4 words.

B. Scalability

We now consider the time of the Streaming algorithm as a function of n , from $n = 5.10^3$ until 2.10^5 . As figure 14 shows, the time grows linearly in n .

C. Error Analysis

We plotted the relative error as a function of n for different values of M , from $M = 2k, 3k, \dots, 7k$, and for each case the error appears to be independent of n , as shown in figure 15.

We made several experiments and increased the value of M to observe a decreased error as shown in figure 16.

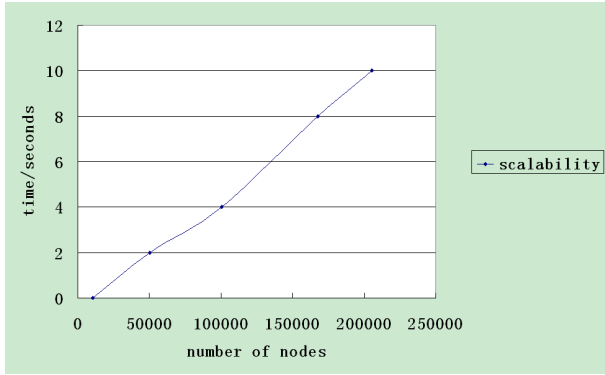


Fig. 14. Time of the Streaming Algorithm as a function of n .

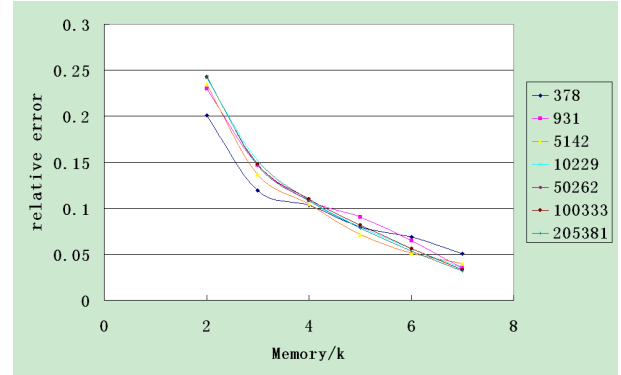


Fig. 16. The relative error as a function of M .

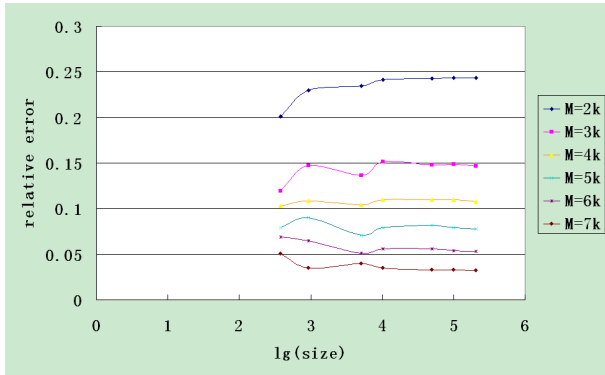


Fig. 15. The relative error as a function of the size n .

D. Conclusion

We described a SAX-implementation of the generalized k -gram associated to an arbitrary unranked ordered tree t . We computed the exact k -gram with a memory d , the depth of the tree, which is too large for a streaming algorithm. The main result of the paper is to show how to approximate the k -gram with a finite memory of size 2^k , the maximal size of the *Tagbuffer*, which is enough to approximately decide the validity of any DTD or Schema. The analysis showed that errors are made only on forks, nodes in the Rabin encoding with two long paths of depth greater than $2k$. We proved that there are few forks and bounded the number of possible errors.

REFERENCES

- [1] E. Fischer, F. Magniez, and M. Rougemont, "Approximate satisfiability and equivalence," in *IEEE Logic in Computer Science*, 2006, pp. 421–430.
- [2] M. de Rougemont and A. Vieilleribière, "Approximate data exchange," in *International Conference on Database Technology (ICDT)*, 2007, pp. 44–58.
- [3] F. Magniez and M. de Rougemont, "Property testing of regular tree languages," in *International Conference on Automata Languages and Programming (ICALP)*, 2004, pp. 932–944.
- [4] P. Flajolet, "Probabilistic counting algorithms for database applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, 1985.
- [5] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–147, 1999.
- [6] L. Segoufin and C. Sirangelo, "Constant-memory validation of streaming xml documents against dtds," in *International Conference on Database Technology (ICDT)*, 2007.
- [7] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse, "The XML Benchmark Project," CWI, Amsterdam, The Netherlands, Tech. Rep. INS-R0103, April 2001.
- [8] R. Rubinfeld and M. Sudan, "Robust characterizations of polynomials with applications to program testing," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 23–32, 1996.
- [9] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," *Journal of the ACM*, vol. 45, no. 4, pp. 653–750, 1998.
- [10] N. Alon, M. Krivelich, I. Newman, and M. Szegedy, "Regular languages are testable with a constant number of queries," *SIAM Journal on Computing*, vol. 30, no. 6, 2000.
- [11] R. J. Parikh, "On context-free languages," *Journal of the ACM*, vol. 13, no. 4, pp. 570–581, 1966.
- [12] A. Broder, "On the resemblance and containment of documents," in *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, 1997.
- [13] G. Cormode and S. Muthukrishnan, "The string edit distance matching problem with moves," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete Algorithms*, 2002.
- [14] M. Garofalakis and A. Kumar, "Xml stream processing using tree-edit distance embeddings," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 279–332, 2005.