Sublinear DTD Validity

Antoine Mbaye Ndione, Aurélien Lemay, Joachim Niehren

Sublinear DTD Validity

Antoine Ndione[1,3,4], Aurelien Lemay[2,3,4], and Joachim Niehren[1,3,4]

Inria Lille[1] & Université de Lille[2] & Cristal Lab[3] & Links Project[4]

Abstract. We present an efficient algorithm for testing approximate DTD validity modulo the strong tree edit distance. Our algorithm inspects XML documents in a probabilistic manner. It detects with high probability the nonvalidity of XML documents with a large fraction of errors, measured in terms of the strong tree edit distance from the DTD. The run time depends polynomially on the depth of the XML document tree but not on its size, so that it is sublinear in most cases (because in practice XML documents tend to be shallow). Therefore, our algorithm can be used to speed up exact DTD validators that run in linear time.

1 Introduction

Validity checking for collections of large XML documents may quickly become time consuming. With today's technology, more than 10 minutes are needed to validate a single document of more than 20 giga bytes, so that the treatment of hundreds such documents may take days or weeks. This difficulty could be overcome by sublinear algorithms that can quickly detect invalid documents without reading them entirely.

Whether sublinear algorithms for XML schema validation exist is a principle question. One approach to obtain sublinear algorithms for schema validation is to use algorithms that evaluate schemas on XML streams in an online manner [9,11]. In this manner, errors can be in sublinear time when they are localized in a prefix of sublinear size of the XML streams, but not otherwise. In contrast, our objective is to develop probabilistic approximation algorithms inspired by property testing [8,7,2] which access a random fragment of constant size only, in order to detect invalidity with high probability, if the input structure contains many errors wheresoever located.

For approximate membership testing for unranked ordered trees as with XML, we need a storage model that permits to randomly draw descendants of any node from a uniform distribution, while giving deterministic access to its first child, next sibling and parent. Such a storage model is easy to implement with the techniques from XML databases. The number of errors is measured by the minimal number of edit operations needed to repair the XML document so that it satisfies the schema, but normalized with respect to the documents size. The more edit operations are permitted, the smaller is the error measured, and the easier becomes approximate membership testing. The only positive result so far applies to testing DTD validity modulo the tree edit distance with subtree moves [7]. But this edit distance is weaker than the usual edit distance, in that

it permits subtree moves beside of all usual operations. Thus the edit distance with move can be very small compared to the usual edit distance and therefore it detects less errors. Approximate membership for tree automata modulo the usual edit distance would be nice to have, but its existence was stated as an open question in [5]. It should also be noticed that property testers for graphs are usually limited to local properties [15,13].

The first contribution of this paper is an approximate membership tester for unranked tree automata modulo the usual tree edit distance [14,18], closing the open problem from [5]. Indeed such a test can be obtained by linearization of unranked trees into words. In order to show this, we use the nontrivial observation that the usual edit distance between two trees is bounded in function of the edit distance of their linearizations [1]. Thereby, can we apply the recent approximate membership tester [12] for non-deterministic finite automata (NFAs) modulo the edit distance on words. This tester improves on a previous tester by Alon, Krivelevich, and Newman [2] for the Hamming distance, so that it runs in polynomial time in the size of the automaton and the inverse error precision, and still independently of the size of the input word. However, the time complexity of the so obtained tester for appoximate membership for unranked tree automata depends exponentially on depth of the input tree. This is not a problem for shallow trees, that are frequent in the case of XML, but leaves open whether this depth dependence can be removed, or whether a polynomial depth dependence can be obtained.

The second and main contribution of this paper is an efficient probabilistic algorithm testing approximate DTD validity modulo the strong tree edit distance from [17], which restricts the usual tree edit operations to leaf insertion, leaf deletion, and node relabeling (while ruling out node inserting and deletion). Its run time depends only on the depth of the XML document but not of its size. Trivially, the same tester is also correct for all weaker distances such as the usual tree edit distance. With inputs: an error precision $\epsilon > 0$, a DTD D, and an XML documents t that is ϵ-far (normalized by the size of t) from satisfying the DTD D modulo the strong edit distance; the algorithm returns NO with high probability. It answers CLOSE for valid trees, and either CLOSE or NO for all others. The running time is polynomially bounded in the depth of t, $1/\epsilon$, and the mintree size m_D of the DTD, which is the maximum over element names $a \in \Sigma$ of the minimal sizes of a-labeled subtrees of D-valid trees. Even though m_D may grow exponentially with D, it seems to be close to the size of D for all practically relevant DTDs. Furthermore, m_D can be computed in quadratic time in the size of D, so unusual cases can be recognized efficiently and passed directly to exact DTD validity checking.

The next difficulty is that we cannot use the linearization approach for approximate membership testing modulo the *strong* tree edit distance. To remedy the situation we study weighted words, i.e., words in which all positions are assigned a weight. The edit distance on words is also lifted to weighted words, such that the costs of edit operations are given by these weights. Then, we extend the algorithm from [12] to a polynomial time NFA membership tester for weighted

words modulo the stong edit distance. We next contribute a direct reduction from approximate DTD validity to approximate NFA membership of weighted words.

Outline. In Section 2 we recall preliminaries on XML data models and schemas. In Section 3, we recall edit distances for trees and words. In Section 4, we present our main result. In Section 5, we introduce weighted words, lift the edit distance, and present our tester for membership of weighted words to regular languages modulo the edit distance. In Section 6, we prove the main result. A long version with full proofs is available at https://hal.inria.fr/hal-00803696.

2 Data Models and Schemas

We recall preliminaries on the XML data model and on XML schemas.

Words. An alphabet Σ is a finite set. We denote the set of words over alphabet Σ by Σ^*. The length of a word $w \in \Sigma^n$ is $|w| = n$ and the set of its positions is $pos(w) = \{1, \dots, n\}$. The empty word is denoted ε and $w \cdot w'$ is the concatenation of w and w'.

A *nondeterministic finite automaton with ε-transitions (*NFA*)* is a tuple $A = (\Sigma, Q, \mathit{init}, \mathit{fin}, \Delta)$, where alphabet Σ is a finite set, Q is a finite set of states with subsets *init* and *fin* of initial and final states, and $\Delta \subseteq Q \times (\Sigma \uplus \{\varepsilon\}) \times Q$ a transition relation.

For states $q,\ q' \in Q$, $\xrightarrow{\varepsilon}$ is the relation such that $q \xrightarrow{\varepsilon} q'$ if and only if $(q, \varepsilon, q') \in \Delta$. In analogy, for any $a \in \Sigma$, $q \xrightarrow{a} q'$ if and only if $(q, a, q') \in \Delta$. The relation $\xrightarrow{\varepsilon}{}^*$ is the reflexive transitive closure of $\xrightarrow{\varepsilon}$. The relation $\overset{a}{\Rightarrow}$ includes multiple ε-transitions and a single a-transition, i.e, $\overset{a}{\Rightarrow}$ is the composition of relations $\xrightarrow{\varepsilon}{}^* \circ \xrightarrow{a} \circ \xrightarrow{\varepsilon}{}^*$.

A *quasi-run* of an NFA A on a word $w = a_1 \dots a_n$ over Σ is a function $r : pos(w) \to Q$ such that $r(i-1) \overset{a_i}{\Rightarrow} r(i)$. A *run* is a quasi-run such that : $\exists q \in \mathit{init},\ q \xrightarrow{\varepsilon}{}^* r(0)$. A run is called *successful* if $r(n) \in \mathit{fin}$. The language $\mathcal{L}(A)$ recognized by A is the set of all words w that permit a successful run.

An NFA $A = (\Sigma, Q, \mathit{init}, \mathit{fin}, \Delta)$ is *productive* if every state in Q is reachable from *init* and co-reachable from *fin*. Without loss of generality we might assume all automata input by our algorithms as productive.

A *fragment* of a word w is a subset of its positions. A fragment of consecutive positions (or without holes) is called an *interval* and denoted by $I =]i, j]$ as usual. A *factor* of w is the word located at an interval, and a *subword* is the word located at a fragment. The subword of w at fragment F is denoted wF.

An interval I of w is called *blocking* for an NFA A if starting from any state of A, every possible quasi-run of A on wI gets stuck, that is, none of those possible quasi-runs occurs in some successful run on a word in $\mathcal{L}(A)$. As an example, if A is an automaton recognizing $L = ab^*$, then the interval $]0, 2]$ of aab is blocking, since after reading the first a, A cannot proceed with any second "a". Whether a fragment is *blocking* is defined similarly, except that the automaton is allowed to jump over holes to arbitrary accessible states. We can decide whether a fragment F is blocking for A in time $O(|F|\ |A|)$ without reading the entire word [12].

XML Data Model. The XML data model essentially boils down to finite unranked data trees when ignoring details of attributes, processing instructions and comments. Since we only consider structural aspects of XML documents described by DTDs, we can safely ignore data values and thus simplify the XML data model further to finite unranked trees over a finite alphabet (fixed by the DTD).

The set of unranked trees over an alphabet Σ is the least set $\mathcal{T}^*_\Sigma$ containing all tuples $a(t_1, \cdots, t_i)$ where $a \in \Sigma$ and $t_1 \cdots t_i$ in $\mathcal{T}^*_\Sigma$. The set $nod(t)$ of nodes of an unranked tree t is a prefix closed subset of $\mathbb{N}^*$ (words with labels in $\mathbb{N}$). The size $|t|$ is the number of nodes of t. As usual, we have the binary relations $parent^t$, fc^t (firstchild) and ns^t (nextsibling) on $nod(t)$. The root of t is denoted by $root^t = root = \varepsilon$ and is the unique node without parent. The i-th child of a node v is $v \cdot i$. A leaf is a node without children. The depth $d(t)$ of a tree is the maximal number of edges on paths from some leaf to the root with parent edges only. For any $v \in nod(t)$ we denote by $t_{|v}$ the subtree of t rooted at node v, and by $t[v] \in \Sigma$ the label of v. Furthermore, we define $word(t) \in \Sigma^*$ to be the sequence of labels of the children of the root of t. For instance, if $t = c(b(a, a), b(a, a, a))$ then $word(t) = bb$ and $word(t_{|root \cdot 2}) = aaa$.

Schemas. Various languages for defining schemas of XML documents were proposed in the literature. Document type descriptors (DTDs) are most basic, while XML SCHEMAs are more expressive. Our choice of DTDs is motivated by the fact that equally efficient membership testers for more expressive formalisms such as tree automata are difficult to find or may even not exist.

Standard DTDs define regular languages of unranked trees by using regular expressions. These can be compiled into NFAs in linear time, but only when permitting ε-transitions as we do [16,10]. It should also be noticed that all regular expressions in DTDs are deterministic (see the W3C recommendation). Therefore they can be converted into deterministic finite automata in polynomial time. However, this conversion might require quadratic time if not fixing the alphabet [4]. For our purpose, it is therefore advantageous to define DTDs based on NFAs with ε-transitions. In some examples we will use regular expressions for illustration nevertheless.

Definition 1. *A* DTD D *over an alphabet* Σ *is a tuple* $(\Sigma, init, (A_a)_{a \in \Sigma})$ *where* $init$ *is an element of* Σ*, and all* A_a *are* NFA*s with alphabet* Σ*.*

An unranked tree t over Σ is valid for a DTD D iff $t[root] = init$ and $word(t_{|v}) \in \mathcal{L}(A_{t[v]})$ for all $v \in nod(t)$. We denote the set of all D-valid trees by $\mathcal{L}(D)$. For all labels $a \in \Sigma$ and DTD $D = (\Sigma, init, (A_a)_{a \in \Sigma})$, we denote by D_a the DTD $(\Sigma, a, (A_a)_{a \in \Sigma})$. The mintree size m_D is the maximum for all $a \in \Sigma$ of all minimal sizes of trees belonging to $\mathcal{L}(D_a)$: $m_D = \max_{a \in \Sigma, \mathcal{L}(D_a) \neq \emptyset} \min\{|t| \mid t \in \mathcal{L}(D_a)\}$. Note that one can compute m_D in quadratic time from D even though this number might be exponentially bigger than the size of D.

3 Edit Distances

We recall the the edit distance for words and trees.

Edit Operations. The (usual) edit operations on words permit to relabel, insert, and delete a letter at a given position. The edit distance between two words w and w' is the least number of usual edit operations needed to transform w into w'. It is denoted by $e(w, w')$. The usual edit operations on trees allow for node relabelling, node inserting, and node deletion [18]. The (usual) edit distance on unranked trees t and t', that we will denote by $e_{stand}(t, t')$ is the least number of usual edit operations required to transform t into t'. The strong edit operations [17] restricts the usual edit operations to node relabelling, leaf insertion and leaf deletion. We consider a tree $t = C(a(t_1, \ldots, t_n))$ where C is a context with hole marker at node v, so that $t_{|v} = a(t_1, \ldots, t_n)$. The relabelling of v to b in t is the tree: $rel_{v,b}(t) = C(b(t_1, \ldots, t_n))$. The insertion of a b-leaf at a position $0 \leq i \leq n$ below node v yields the tree: $ins_{v,i,b}(t) = C(a(t_1, \ldots, t_i, b, t_{i+1}, \ldots, t_n))$. The deletion of a leaf $v \cdot i$ with $1 \leq i \leq n$ yields: $del_{v \cdot i}(t) = C(a(t_1, \ldots t_{i-1}, t_{i+1}, \ldots, t_n))$. The strong edit distance between two trees t and t' is the least number of strong edit operation to transform t into t'. It is denoted by $e(t, t')$.

It always holds that $e_{stand}(t, t') \leq e(t, t')$. Furthermore, $e(t, t') \leq |t| + |t'| - 1$ since we can first delete all nodes of t except the root, then relabel the root, and finally add all non-root nodes of t' one by one.

Farness. Let S be a set of structures and $e : S \times S \to \mathbb{N}_0$ a function called the distance for S. We assume that any structure $s \in S$ has a finite size $|s| \in \mathbb{N}_0$. We define the distance of a structure s to a language $L \subseteq S$ as the least number of edit operations needed to transform s into a member of L, i.e., $E(s, L) = \min_{s' \in L} E(s, s')$.

Definition 2. *Let $\epsilon > 0$ and $\mathcal{L}(A) \subseteq S$ for some language definition A. A structure s is called ϵ-far from A modulo distance E if the normalized distance of s to $\mathcal{L}(A)$ is greater than ϵ, that is if $E(s, \mathcal{L}(A))/|s| \geq \epsilon$, and ϵ-close otherwise.*

Note that ϵ-farness from a DTD D with respect to the usual edit distance implies ϵ-farness from D with respect to the strong edit distance. Furthermore, since $e(t, t') \leq |t| + |t'| - 1$ for any two unranked trees, it follows that $e(t, D_a) \leq |t| + m_D - 1 \leq m_D\ |t|$ for all labels a in the alphabet of a non-empty DTD D (a tree always has a root, so $|t|, m_D \geq 1$). Since emptiness of DTDs is linearly decidable we only consider non empty DTDs in the rest of the paper.

Linearization. The relationship from [1] between the usual edit distance on trees t and t' and the edit distance of their respective XML linearizations w and w' depends on the minimal depth of the two trees d:

$$\frac{e(w, w')}{2} \ \leq\ e_{stand}(t, t') \leq (2d + 1)\ e(w, w')$$

These estimations are thight up to a constant factor. For any tree t of depth d, if t is ϵ-far from a DTD D modulo the usual tree edit distance, then its XML linearization w is $\frac{\epsilon}{2d+1}$-far from the linearizations of any D-valid tree.

Note however, that the same upper bound does not hold for the strong tree edit distance $e(t, t')$. This indicates already, that we will need a more general method for testing DTD membership modulo the strong tree edit distance, than for testing membership for finite word automata modulo the edit distance.

4 Main Results

Sublinear membership testers are not allowed to read the whole input structure. Instead they only access some elements of the structure randomly and navigate from there on. Which access operations are permitted can be defined by a randomized data model. In this section, we introduce appropriate randomized data models for words and trees, and then formulate our positive results.

Randomized Data Models. As usual, any word w with alphabet Σ defines a unique relational structure S_w with domain $dom(w) = pos(w) \cup \{0\}$, that is $S_w = (dom(w), start^w, succ^w, (lab_a^w)_{a\in\Sigma})$ where $start^w = \{0\}$, $succ^w = \{(i, i+1) \mid 0 \leq i < |w|\}$, and lab_a^w is the set of positions of w labeled by a. The randomized data model is similar except that it gives random access to elements of some structure isomorphic to S_w. More formally, the randomized data model of a word w and a bijection $\theta : dom(w) \to V$ is the tuple $rdm_w^\theta = (ele, start, succ, lab)$, which contains a random generator ele that draws an arbitrary element of V from a uniform distribution, a start position $start \in V$, a successor function $succ : V \to V \cup \{\bot\}$, and a labeling function $lab : V \to \Sigma$, such that θ is an isomorphism between S_w and the relational structure $(V, \{start\}, \{(v, succ(v)) \mid succ(v) \neq \bot\}, (\theta(lab_a^w))_{a\in\Sigma})$.

Any XML document t, as an unranked tree over some alphabet Σ, defines a unique relational structure $S_t = (nod(t), root^t, parent^t, fc^t, ns^t, (lab_a^t)_{a\in\Sigma})$ with domain $nod(t)$. The randomized data model is similar except that it gives random access to elements of some structure isomorphic to S_t. More formally, the randomized data model of a word w and a bijection $\theta : nod(t) \to V$ is the tuple $rdm_t^\theta = (desc, root, parent, fc, ns, lab, depth)$. For any node v of t, $desc(v)$ is a random generator that draws descendants of v from a uniform distribution or returns $\bot$ if v is a leaf, a root element $root \in V$, a parent function $parent : V \setminus \{root\} \to V$, the firstchild function $fc : V \to V \cup \{\bot\}$, the nextsibling function $ns : V \to V \cup \{\bot\}$, and labeling functions $lab : V \to \Sigma$. We require that θ is an isomorphism from S_t to the relational structure $(V, \{root\}, \{(v, parent(v)) \mid v \in V\}, \{(v, fc(v)) \mid fc(v) \neq \bot\}, \{(v, ns(v)) \mid ns(v) \neq \bot\}, (\theta(lab_a^t))_{a\in\Sigma})$. Finally, $depth = d(t)$ is the depth of t.

Approximate Membership. Approximate membership is a special case of property testing, aiming for probabilistic algorithms that read a sublinear part of the input structure based on a randomized data model. For the formal definition, we fix a class $\mathcal{S}$ of structures such that each structure s in $\mathcal{S}$ has randomized data models denoted by rdm_s, and a class $\mathcal{A}$ of language definitions such that each definition A in $\mathcal{A}$ defines a language $\mathrm{L}A$ included in $\mathcal{S}$ and has a size $|A|$ in $\mathbb{N}$.

Definition 3. *An approximate membership tester for $\mathcal{S}$ and $\mathcal{A}$ is an algorithm (possibly randomized) that receives as inputs a randomized data model rdm_s for some structure $s \in \mathcal{S}$, an error precision $\epsilon > 0$ and a language definition $A \in \mathcal{A}$, and answers with probability $\frac{2}{3}$: CLOSE if $s \in \mathcal{L}(A)$ and NO if s is ϵ-far from A.*

The *query complexity* of a tester is the number of times it uses rdm_s during the computation in dependence of the input (size). Its *time complexity* accounts

for all other operation performed by the algorithm in addition to the query complexity.

Tree Edit Distance. We next sketch how to test approximate membership for tree automata on unranked trees [6] modulo the usual tree edit distance, based on tree linearization. Note that such tree automata subsume our DTDs.

The idea is to use the upper bound $e_{stand}(t,t') \leq (2d+1)e(w,w')$ from [1], where w is the XML linearization of t and w' the XML linearization of t'. We want to test approximately whether an unranked tree t of depth d is recognized by a tree automaton B. If t is ϵ-far from B modulo the usual tree edit distance, then its linearization is $\frac{\epsilon}{2d+1}$-far from the language of linearizations of trees recognized by B of depth at most d. The tree automaton B can then be compiled into finite automata A of exponential size $|B|^d$ that accepts all these linearizations. This can be done by first compiling B into a nested word automaton [3] in linear time, which in turn is compiled to a finite automaton by moving stacks up to depth d into states. One can then apply the polynomial time membership tester for finite automata modulo the edit distance on words from [12]. In order to do so, one has to verify that the randomized data model of words can be simulated by a randomized data model of the corresponding tree, which is straigthforward. Since the tester in [12] never errs for correct words, and there is no requirement on close trees, this method gives indeed a valid tester. The query complexity of this test is in $O(p(|A|, 1/\epsilon, d))$ where p is the polynomially bounded function that satisfies for all positive real numbers $\mathbf{a}, \mathbf{e}, \mathbf{d}$: $p(\mathbf{a}, \mathbf{e}, \mathbf{d}) = \mathbf{a}^3\ \mathbf{e}\ \mathbf{d}\ \log^3(\mathbf{a}^2\ \mathbf{e}\ \mathbf{d})$ The time complexity is in $O(|A|\ p(|A|, 1/\epsilon, d))$. In combination we obtain:

Theorem 4. *Whether an unranked tree t is approximatively recognized by a tree automaton B can be tested with query complexity and time complexity in $O(p(|B|^{d(t)}, 1/\epsilon, d(t)))$ and $O(|B|^{d(t)}\ p(|B|^{d(t)}, 1/\epsilon, d(t)))$ respectively; modulo the tree edit distance with error precision ϵ*

Even though nontrivial, this theorem has three weaknesses. First of all, the finite automaton A constructed from the tree automaton B and the depth d may be of exponential size $O(|B|^d)$. Second, the tester does not apply to the strong tree edit distance. And third, the query complexity of the tester depends exponentially on the depth of the tree.

Strong Tree Edit Distance. Our main result is that all three problems can be solved for DTD membership modulo the strong tree edit distance, as stated in the following theorem.

Theorem 5. *Whether an unranked tree t is valid for a* DTD $D = (\Sigma, init, (A_a)_{a\in\Sigma})$ *modulo the strong tree edit distance with error precision ϵ can be tested with query complexity in $O(d^2\ p(\boldsymbol{a}, d/\epsilon, m_D))$ and time complexity in $O(\boldsymbol{a}\ d^2\ p(\boldsymbol{a}, d/\epsilon, m_D) + |D|)$, where $d = d(t)$ and $\boldsymbol{a} = \max_{a\in\Sigma} |A_a|$ is smaller than $|D|$.*

The dependency on the depth is reduced from exponential to polynomial. In contrast, approximate membership of NFAs can be done with constant query complexity [12,2,7]. Nevertheless, as DTDs are naturally connected to NFAs, one might want to reduce approximate membership of the former to the one of

the latter. We believe that this cannot be archieved. Instead, we will present a reduction to a more general property tester for so called weighted words that we will develop for this purpose.

5 Weighted Words

We present an approximate membership tester for finite automata on weighted words modulo a weighted edit distance.

From Trees to Weighted Words. A weighted word over an alphabet Σ is a word over the alphabet $\Sigma \times \mathbb{N}$. The idea for the introduction of weigthed words is as follows. To any node v of a tree t we assign the weight $|t_{|v}|$. The weighted word associated to a node v is then the word $word(t_{|v})$, in which each position is weighted by the weight of the corresponding child of v.

We next illustrate the close link from trees to weighted words by example. We consider the DTD D with rules $r \to ab^*, a \to a^*, b \to b^*$. For any $i \geq 0$, let a_i be the tree $a(a, \ldots, a)$ with i a-leaves and b_i the tree $b(b, \ldots, b)$ with i b-leaves. The tree $t = r(a_1, b_2, b_3, a_4)$ of depth 2 is clearly invalid for D. Its distance is $e(t, D) = 5$ since one must delete the whole last subtree to become valid and this subtree has size 5. However, if we consider the regular language below the root $L = ab^*$ and pick the word at the root $w = word(t_{|\varepsilon}) = abba$, then we have $e(w, L) = 1$ for the edit distance for words. One way to understand the problem is that we cannot simply ignore the sizes of the subtrees as we did. Instead, we should associate a weight to each position, and consider the weighted word $\omega = (a, 2)(b, 3)(b, 4)(a, 5)$ for the above example. We also need to adapt the costs of deleting a weighted letter such as (c, i) to its weight i. In this way, the weighted distance of ω to L becomes 5 which is equal to the distance of t to D.

Any weighted word has the form $w * p$ for some $w \in \Sigma^*$ and $p \in \mathbb{N}^*$, where w and p have the same length. We call w the word part and p the weight part of $w * p$. We will also say that ω has at position i the weight $k \in \mathbb{N}$ and the label $a \in \Sigma$ if $\omega[i] = (a, k)$. The weight $|\omega|_*$ is the sum of the weights at all positions of ω. The word part of a weighted word is used to define its membership to word regular languages while the weight part is used to define weighted words edit distance. We say that a weighted word $w * p$ is recognized by an NFA A if and only if $w \in \mathcal{L}(A)$. The set of weighted words recognized by A is denoted by $\mathcal{L}_*(A)$. The notions of blocking fragment and interval are lifted to weighted words by deletion of the weights. For example, if A is a productive automaton recognizing $L = ab^*$, then the interval $I =]0, 2]$ of $\omega = (a, 1)(a, 3)(b, 4)$ is blocking for A, since aa is the word part of the weighted word ωI located at I, and after reading the first a, A cannot proceed with any second "a".

The edit operations for weighted words are essentially the same as for words, i.e, insertions, relabeling, and deletions. The only difference is that the costs of these operations depend on the weights of the letters that are edited. For a weighted word $\omega = \sigma_1 \ldots \sigma_n$ and a natural number $i \in [0, n]$, the insertion of a weighted letter $\sigma \in \Sigma \times \mathbb{N}$ following position i in ω yields the weighted word: $ins_{i,\sigma}(\omega) = \sigma_1 \ldots \sigma_i \sigma \sigma_{i+1} \ldots \sigma_n$. The cost of this insertion is the weight

of σ. The deletion of position $1 \leq i \leq n$ of w yields the following weighted word: $del_i(\omega) = \sigma_1 \dots \sigma_{i-1}\sigma_{i+1} \dots \sigma_n$. The cost of such a deletion operation is the weight of the deleted letter σ_i. The relabeling at position $1 \leq i \leq n$ of ω into a letter b changes only the letter at this position but not its weight. Let $\sigma_i = (a, k)$ then the relabeling operation at position i costs k and yields: $rel_{i,b}(\omega) = \sigma_1 \dots \sigma_{i-1}(b, k)\sigma_{i+1} \dots \sigma_n$.

Testing Weighted Words. We next show that approximate NFA membership modulo the edit distance can be tested efficiently for weighted words. We will prove the following result for the randomized data model of weighted words defined below.

Theorem 6. *Let A be an NFAs that has k strongly connected components. Whether a weighted word ω is approximately a member of $\mathcal{L}_*(A)$ modulo the weighted edit distance with error precision ϵ can be tested with query complexity $O(\frac{k^2|A|}{\epsilon}\log^3(\frac{k|A|}{\epsilon}))$ and time complexity $O(\frac{k^2|A|^2}{\epsilon}\log^3(\frac{k|A|}{\epsilon}))$ independently of the weight or size of ω.*

So far, the ideas to Theorem 6 are essentially the same as for usual words [12]. What changes for weighted words is that many errors can be concentrated at some position of high weight. This can be accounted by adapting the random drawing of fragments. Instead of using a generator that draws positions uniformly in the word, we use a random generator that draws positions depending on weights. The probability to draw the i-th position of a weighted word $\omega = w * p$ should be $p(i)/|\omega|_*$. We call such a random generator a drawing from a weighted distribution. We can now define the random data model of a weighted word ω and a bijection $\theta : pos(\omega) \to V$ in analogy to the case of words: $rdm^\theta_\omega = (ele, start, next, lab)$. Here, *ele* is a random generator of positions for the weighted distribution. It might be disturbing that such random generator cannot be obtained from a weighted word without reading it entirely. However, as we will see, we can obtain it from the randomized data model of a tree.

With respect to such random data models for weighted words, Theorem 6 become true. We prove this in two steps. We first consider the case where the NFA is strongly connected, and second study the general case of automaton with several strongly connected components.

Strongly Connected Automata. Let A be an NFA that is strongly connected. An approximate membership testing for weighted words can proceed as follows. The input is a randomized data model rdm^θ_ω for some weighted word ω. The tester then generates randomly sufficiently many positions of the word according to their weights, reads sufficiently long factors starting there, and returns NO if one of them is blocking. What "sufficient" here means can be deduced from the following Lemma.

Lemma 7. *Let $A = (\Sigma, Q, init, fin, \Delta)$ be an NFA that is strongly connected, ω a weighted word, m a natural number bigger than $|\omega|_*$, $\epsilon > 0$ an error precision, and $\gamma = \frac{8|Q|}{\epsilon}$. If $e(\omega, A) > \epsilon m$ and $m \geq 8\gamma\lceil\log(\gamma)\rceil$, then there is a length $l \in [2, \gamma]$ which is a power of 2, and a set of disjoint intervals $\mathcal{I}_l$ with weight $m\beta_l$ such that: all intervals of $\mathcal{I}_l$ are of lentgh $2l$ and blocking for A. Where $\beta_l = l/(2\gamma\lceil\log(\gamma)\rceil)$.*

```
fun member_A(r, ε)
  // r = rdm_ω^θ for some weighted word ω
  // and ε an error precision
  let (ele, start, succ, lab) = r
  let k be the number of strongly connected components of A
  let γ' = 16k|Q|/ε
  if |ω| < 8γ'⌈log(γ')⌉ then
     if ω ∈ L(A)//run A via start, succ, lab
     then return CLOSE else return NO
  else
     for i = 1 to ⌈log(γ')⌉ do
       let l = min(2^i, γ')
       let α_l = 30kγ'⌈log(γ')⌉^2/l
       let S be sequence of α_l positions of ω randomly drawn by ele
       let F be the union of all intervals of ω of length 2l starting
           at positions in S
       if F is blocking wrt. A
          // run A via succ and lab
       then return NO; exit else skip
     end
     return CLOSE
```

Fig. 1. An approximate membership tester for weighted words.

General Automata. We generalised the previous result to automata with multiple strongly connected components. We refine the results from [12], which in turn adapts the schema from [2], and prove that drawing positions from the weight distribution of ϵ-far weighted words, yields a blocking fragment with high probability.

For integers l, α, weighted word ω and a sequence $S = (i_1, \cdots, i_\alpha)$ of α positions in ω, we denote by F_S the fragment $\cup_{1 \leq j \leq \alpha}[i_j, \min(i_j + l, |\omega|)]$. And we define $\mathcal{S}(\omega, l, \alpha)$ as the set of all fragments F_S.

Lemma 8. *Let A be a productive* NFA *with state set Q and k strongly connected components. Let $\epsilon > 0$, $\gamma' = \frac{16k|Q|}{\epsilon}$ and ω be a weighted word of weight greater than $8\gamma'\lceil\log(\gamma')\rceil$. If ω is ϵ-far from A, then there exists a power of two $l \in [2, \gamma']$ such that: with probability $\frac{5}{6}$, drawing $\alpha_l = 30k\gamma'\lceil\log(\gamma')\rceil^2/l$ positions with the weight distribution of ω yields some blocking fragment in $\mathcal{S}(\omega, 2l, \alpha_l)$.*

Finally, Theorem 6 is a consequence of Lemma 8. Indeed, the algorithm in Figure 1 is a one sided membership tester for weighted words. In fact by the previous lemma, drawing enough positions according to the weight distribution gives a blocking fragment with probability at least $\frac{5}{6} \geq \frac{2}{3}$. The case of weighted words with small lengths is easily detected using *start*, *succ*, *lab* and the exact membership is checked; this case includes all light weighted words. Furthermore weighted words in $\mathcal{L}(A)$ have no blocking fragments.

6 Testing Unranked Trees

We reduce DTD membership of trees to NFA membership of weighted words. For a tree t, the reduction is based on the weighted words $\omega_v = word(t_{|v}) * p_v$ for

nodes v of t, where p_v is the sequence of sizes $|t_{|v'}|$ of subtrees rooted at the children v' of v in document order. Note that we do not need to compute the values $|t_{|v'}|$. However, we can draw a child v' of some node v in a tree t with probability $|t_{|v'}|/|t_{|v}|$, and this is the only thing needed by our weighted word tester (Figure 1), as this drawing corresponds to the drawing of positions for the weighted distribution of ω_v. Therefore, by Theorem 6, for all nodes v of t, we can test membership of ω_v to NFAs efficiently. However, the query complexity measured in terms of accesses to tree t belongs only to $O(d(t) \cdot \frac{k^2|A|}{\epsilon} \log^3(\frac{k|A|}{\epsilon}))$, since we need to draw children of v as explained above in order to draw positions in ω_v with the correct weight.

We now link the strong tree edit distance to DTDs to the edit distance of weighted words to NFAs.

Lemma 9. *Let $D = (\Sigma, init, (A_a)_{a\in\Sigma})$ be a* DTD*, $\epsilon > 0$ a precision, and t a tree. If all nodes $v \in nod(t)$ satisfy $e(\omega_v, A_{t[v]}) \leq \frac{\epsilon}{m_D}|\omega_v|_*$ and $lab(root^t) = init$ then $e(t, D) \leq d(t)\epsilon|t|$.*

Lemma 9 shows that trees ϵ-farness is witnessed by nodes with weighted words far from their appropriate regular language. We next explain how to detect such nodes. Indeed the next lemma states that the overall subtree sizes of nodes whose corresponding weighted words are far from their regular word language is important. A node $v \in nod(t)$ is ϵ-bad if $e(\omega_v, A_{t_{|v}}) > \frac{\epsilon}{2m_D \cdot d(t)}|\omega_v|_*$. Let B_t be the set of bad nodes whose ancestors aren't bad. $|B_t| = \sum_{v\in B_t} |t_{|v}|$ is the size of B_t.

Lemma 10. *For* DTD *D, precision $\epsilon > 0$, and t a tree ϵ-far from D, one has $|B_t| > \frac{\epsilon}{2}|t|$.*

We describe now how this lemma translates to a membership tester. Let μ be the random process that uniformly selects a node of t and returns its path to the root π. The size of π is at most $d(t)$ and for all nodes $v \in nod(t)$, the probability that v is in π is $Pr_{\pi\sim\mu}[v \in \pi] = |t_{|v}|/|t|$. Therefore, by Lemma 10, if t is ϵ-far from D, then the probability that π contains an element of B_t is at least $\frac{\epsilon}{2}$. Hence for $\lceil \log(5)/\epsilon \rceil$ drawn paths, with probability $\frac{4}{5}$ one drawn node is bad. However we do not know which selected node is bad, so we need to verify them all to detect ϵ-farness. We next use the membership tester of weighted words in section 5, which answers correctly with probability at least $\frac{5}{6}$. It follows that with probability at least $\frac{4}{5} \cdot \frac{5}{6} = \frac{2}{3}$ we would find error in t.

Conclusion and Future Work. We have presented the first approximate membership tester for DTDs modulo the strong tree edit distance. The most difficult part was to extend previous results for regular words languages to regular tree languages that are restricted to locality in vertical direction (but not horizontally). Some questions remain open. First of all, it might be possible that approximate membership modulo the edit distance can be tested efficiently for XML SCHEMAs by extending the methods presented here. In such a setting one would preserve top-down determinism but give up vertical locality. A second more

difficult question is whether approximate membership can be tested efficiently for bottom-up tree automata for ranked trees, while depending only on their depth. The third yet more difficult question is whether efficient algorithms exist for testing RelaxNG validity. Fourth, it might be interesting to study property testing for schemas with key constraints.

References

1. Akutsu, T.: A relation between edit distance for ordered trees and edit distance for Euler strings. Inf. Process. Lett. pp. 105–109 (2006)
2. Alon, N., Krivelevich, M., Newman, I., Szegedy, M.: Regular Languages are Testable with a Constant Number of Queries. SIAM J. Comput. pp. 1842–1862 (2000)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM pp. 1–43 (2009)
4. Brüggemann-Klein, A.: Regular Expressions to Finite Automata. Theoretical Computer Science pp. 197–213 (1993)
5. Chockler, H., Kupferman, O.: w-Regular languages are testable with a constant number of queries. Theor. Comput. Sci. pp. 71–92 (2004)
6. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007)
7. Fischer, E., Magniez, F., de Rougemont, M.: Approximate Satisfiability and Equivalence. In: LICS. pp. 421–430 (2006)
8. Goldreich, O.: Combinatorial Property Testing (a survey). In: In: Randomization Methods in Algorithm Design. pp. 45–60 (1998)
9. Green, T.J., Gupta, A., Miklau, G., Onizuka, M., Suciu, D.: Processing XML streams with deterministic automata and stream indexes. ACM Trans. Database Syst. pp. 752–788 (2004)
10. Hagenah, C., Muscholl, A.: Computing epsilon-free nfa from regular expressions in $O(nlog^2(n))$ time. ITA pp. 257–278 (2000)
11. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML schema. ACM Transactions of Database Systems pp. 770–813 (2006)
12. Ndione, A., Lemay, A., Niehren, J.: Approximate membership for regular languages modulo the edit distance. Theor. Comput. Sci. pp. 37–49 (2013)
13. Newman, I., Sohler, C.: Every property of hyperfinite graphs is testable. In: STOC. pp. 675–684 (2011)
14. Pawlik, M., Augsten, N.: RTED: A Robust Algorithm for the Tree Edit Distance. PVLDB pp. 334–345 (2011)
15. Ron, D.: Property Testing: A Learning Theory Perspective. Foundations and Trends in Machine Learning pp. 307–402 (2008)
16. Schnitger, G.: Regular expressions and nfas without *epsilon*-transitions. In: STACS. pp. 432–443 (2006)
17. Selkow, S.M.: The Tree-to-Tree Editing Problem. Inf. Process. Lett. pp. 184–186 (1977)
18. Zhang, K., Shasha, D.: Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. SIAM J. Comput. pp. 1245–1262 (1989)