

# Correctors for ranking XML documents

Utsav Boobna  
I.I.T Kanpur  
India  
utsav@iitk.ac.in

Michel de Rougemont  
University Paris II  
LRI Batiment 490  
91405 Orsay, France  
mdr@lri.fr

## ABSTRACT

A corrector takes an invalid XML file  $F$  as input and produces a valid file  $F'$  which is not far from  $F$  when  $F$  is  $\epsilon$ -close to its DTD, using the classical Tree Edit distance between a tree  $T$  and a language  $L$  defined by a DTD or a tree-automaton. We show how testers and correctors for regular trees can be used to rank XML documents.

We describe the implementation of linear time correctors using the Xerces parser and present test data for various DTDs comparing the parsing and correction time. We propose a generalization to homomorphic DTDs.

## 1. INTRODUCTION

For classification, search and querying semistructured data, it is important to detect approximate validity, i.e. if a file  $F$  approximately follows a structure  $M$ . In the case of XML data, the structure is a DTD or schema and we want to decide if a file  $F$  approximately follows a DTD  $M$ .

A given file is well-formed if the tags follow a tree-like structure and is valid if the tree is accepted by the automaton associated with the DTD. The tool Tidy [13] takes an XML file as input and corrects it if it is not well-formed: it adds the missing tags and removes the incorrect ones to obtain a well-formed file close to the original one. We describe the implementation of a similar tool for validity. It takes a well-formed XML file and a DTD as input and corrects it if it is invalid: it adds the missing leaves and subtrees, removes or modifies the incorrect ones to obtain a valid file close to the original one. Moreover it estimates in linear time the distance between a file and several DTDs which can be used to *rank documents* relative to these DTDs.

The distance between two XML files is the classical Tree Edit Distance [14, 12, 2] applied to the DOM trees associated with the files. It measures the number of insertions and deletions of nodes and edges, and the number of label changes to a given node. It generalizes the classical Edit Dis-

tance on strings which measures the number of insertions, deletions and modifications of characters necessary to apply to a string  $s$  to obtain a string  $s'$ . These Edit Distances have many extensions when additional operators are used, such as Moves of substrings, Permutations and Cut/Paste operations.

Correctors are related to the self-testers of [3, 4] in the theory of program verification. The related notion of *property testing* was proposed in [11], its applications to graphs were studied in [8] and its applications to regular words in [1]. These last authors prove that regular properties of words have testers, i.e. we can decide by sampling a word in constant time, if it belongs to a regular language or if it is far from the language, i.e. at distance more than  $\epsilon \cdot n$  if  $n$  is the length of the string. An XML tester for a DTD and some  $\epsilon$  takes an XML file  $F$  and decides with high probability in constant time if  $F$  is valid or if  $F$  is  $\epsilon$ -far from the DTD. In [9], we prove the existence of such a tester for regular trees and the Tree Edit Distance with moves.

A corrector for regular words transforms a word  $s$  close to a regular language, i.e. at a distance less than  $\epsilon \cdot n$  into a word  $s'$  in the language such that  $s'$  is close to  $s$ . For regular tree properties, i.e. properties defined by tree automata, or by DTDs, similar testers and correctors exist. A basic corrector was introduced in [7]. It is a linear algorithm which takes a general invalid file  $F$  and a DTD  $M$  as input and produces a valid file  $F'$  close to  $F$  as output, when the distance between  $F$  and the DTD is constant, i.e. if there are not too many errors. It introduced the notion of a *global correction* which implied a time linear in  $n$  but exponential in the number of errors.

In this paper, we first give a global presentation of correctors and describe the implementation of a *local corrector* for the Tree Edit distance without moves, i.e. an algorithm which is both linear in  $n$  and the number of errors but may not yield the best correction. We describe its implementation using the Xerces parser and give test data for invalid files which follow three DTDs: the first represents deep trees, the second wide trees, and the third a combination of deep and wide trees. The main result of the paper is to show that the correction time is always less than the parsing time and therefore scales up.

We discuss the generalization of a corrector to close DTDs. We consider a file  $(F, M_1)$  where  $M_1$  is the DTD of  $F$ , and

an external DTD  $M_2$  close to  $M_1$ , having defined the distance between two DTDs. We estimate the distance between  $(F, M_1)$  and  $M_2$ , when  $F$  is close to  $M_1$ ,  $M_2$  close to  $M_1$  and therefore  $F$  is close to  $M_2$ .

In section 2, we present the notations, the Tree Edit Distance, property testing on trees and its connection with the correctors. In section 3, we describe the structure of correctors, the implementation of a local corrector and give an example. In section 4, we provide general figures on the relative parsing and correction time for three DTDs. In section 5, we introduce a distance on DTDs and generalize the approach to homomorphic DTDs.

## 2. PRELIMINARIES

A well-formed XML file is composed of two parts: a *ranked ordered labelled tree* and a Document Type Definition DTD or Schema. We define these two objects, the Tree Edit Distance which gives a measure between ordered labelled trees and between a tree and a DTD, and the basic notions of testers and correctors.

### 2.1 Ranked ordered labelled trees

A *ranked labelled ordered tree* is a ranked ordered tree with labels, i.e. a structure  $\mathcal{T}_{ri} = (D_n, Child_i, Label_j, root)_{i \leq m, j \leq p}$  where the domain  $D_n = \{1, \dots, n\}$  is the set of nodes, the binary relation  $Child_i(u, v)$  is satisfied if  $u$  is the  $i$ -th child of  $v$  for  $i \leq m$  and a fixed  $m$  and  $root$  is a distinguished element of  $D_n$  with no predecessors. The graph of the  $Child_i(u, v)$  is a tree,  $Label_j$  is a unary relation on  $D_n$  and the set  $\{Label_j\}_{1 \leq j \leq p}$  is a partition of  $D_n$ .

Let  $\mathcal{L} = \{l_1, l_2 \dots l_j \dots l_p\}$  be the set of labels also called *tags*. A DTD defines one label as a *root* and declares a set of rules  $l : (m)$  where  $l$  is a tag and  $m$  a regular expression on  $\mathcal{L}$ , also called the *content model*  $m$ . The content model ( $\#PCDATA$ ) indicates a leaf. Each rule  $l : (m)$  specifies a transition of an unranked tree-automaton. In this paper we consider bottom-up tree automata first on labelled trees but we may have taken equivalent models. In the case of real DTDs, we can adapt to one-unambiguous languages and obtain an effective generalization of parsers for XML schemas.

Our implementation uses the standard parsing tools, Sax and Xerces. After reading the file, we obtain a DOM tree, which is parsed bottom-up. We describe how to handle parsing errors, in order to modify a DOM tree and obtain a valid one. We interleave parsing steps with correction steps and obtain two outputs: a corrected file and an estimation of the Tree Edit Distance.

### 2.2 The Tree Edit Distance

The Edit Distance on strings has been introduced in [14] for comparing strings and generalized in [12] for trees. The survey [2] describes hardness results for unordered trees and classical polynomial algorithms for ordered trees. Basic operations on ordered labelled trees include: *change* a label, *insertion* of a node on a given edge (transformed in two edges), *insertion* of an edge or *deletion* of an edge.

DEFINITION 1. *The distance between  $T$  and  $T'$  is the minimum number of Tree Edit operations necessary to reach  $T'$*

from  $T$ , noted  $Dist(T, T')$ . The distance between  $T$  and a language  $L$ , noted  $Dist(T, L)$  is the minimum distance between  $Dist(T, T')$  for  $T' \in L$

We say that two trees  $T$  and  $T'$  are  $k$ -close if their distance is less than  $k$ , and that  $T$  is  $\epsilon$ -close to a DTD if the distance between  $T$  and the language  $L$  associated with the DTD is less than  $\epsilon \cdot n$ , if  $T$  has  $n$  nodes.  $T$  is  $\epsilon$ -far from a DTD if it is not  $\epsilon$ -close. It is a classical observation that the distance is P-computable for ordered trees and NP-complete on unordered trees. In the context of XML, several papers [10, 5] estimate similarities between files using variations of the P-algorithm based on dynamic programming, an  $O(n^2)$  algorithm. The methods we introduce can estimate distances to a DTD in constant time if we use a tester and in linear time if we use a corrector which also produces a corrected file.

#### 2.2.1 Distances with moves

On strings, a *move* is simply the possibility to isolate an arbitrary substring  $t$  and a position  $i$  in a string  $s$  of size  $n$  and to *move*  $t$  in position  $i$ , shifting the word to the right in one step. The new Edit distance, called *Edit distance with moves* introduced in [6] has many applications in the database streaming model.

In the case of trees, a *move* is the possibility to isolate a subtree  $t$  and a leaf  $i$  in a tree  $T$  of size  $n$ . We move  $t$  to position  $i$  in one step. This construction is indeed feasible in the DOM tree model by modifying two edges.

PROPOSITION 1. *The distance problem on ordered trees with the Edit distance with moves is NP-complete.*

A proof is given in [9] and relies on the fact that ordered trees with moves are equivalent to unordered trees, from the distance point of view. Subtrees below a node can be switched with 2 moves. Although this distance is hard to compute, it becomes easy to approximate in the sense introduced by testers.

### 2.3 Testers and Correctors

In the late eighties, the theory of *program checking* and *self-testing/correcting* was initiated in [3, 4]. Correctors in this context take a program, incorrect on a few instances, as an oracle and produce a correct program with high probability. Many interesting correctors for numerical computations and linear algebra are presented in [3].

The related notion of *property testing* was first proposed in [11]: it is an approximate randomized test which separates with high probability inputs which satisfy a property from those which are far from the property, using the Hamming distance to compare structures. For graphs, [8] investigated property testing for several properties such as  $c$ -colorability which can be approximately tested in constant time. In [1], the authors prove that any regular property of strings can be tested and this result is generalized for the Edit Distance in [9]. For trees with the Tree Edit distance with moves, [9] provides a randomized algorithm  $A$  which given a tree  $T$  of size  $n$  and a regular tree language  $L$  defined by a DTD is such that:  $A$  accepts if  $T \in L$  and  $A$  rejects with high

probability if  $T$  is  $\epsilon$ -far from  $L$ , i.e. the distance between  $T$  and  $L$  is greater than  $\epsilon \cdot n$ . The time of the algorithm is independent of  $n$  and depends on  $\epsilon$  only. Such an algorithm samples the DOM tree by selecting a random node and a finite local subtree of size  $1/\epsilon$ , and finally checks a local property.

There is a corrector which takes a tree  $T$   $\epsilon$ -close to  $L$ , i.e. at distance less than  $\epsilon \cdot n$  as input and produces a tree  $T' \in L$ , which is  $\epsilon'$ -close to  $T$ , as output.

Notice that the Tree Edit distance with moves is hard to compute but we can estimate it if it is small. First use the tester to detect if it is large. If it is small, use a corrector for an estimate.

### 3. CORRECTORS FOR XML

A Corrector takes an XML file (which may not be valid) and the corresponding DTD as input and produces a valid XML file close to the original one and the approximate Edit distance, as output. For the Tree Edit distance without moves, we don't know correctors which would correct up to distances  $\epsilon \cdot n$  but we know correctors which correct up to a constant distance. We now present two classes of correctors and an implementation for this distance.

#### 3.1 Corrector for the Tree Edit distance

There are two distinct steps proposed in [7], after having read a file.

1. **Inductive \*-Marking of the DOM tree.** Follow a bottom-up run on the DOM tree and mark with a \* the nodes where a parsing error occurs. The root of a subtree which contains \*-nodes is a \* node if all substitutions of \* with feasible tags lead to a parsing error.

2. **Recursive correction of the \*-subtrees.** Proceed top-down and propose local modifications in the neighborhood of the \* node of maximum height. Compute the global distance obtained when the leaves are reached and choose the modification of minimum distance.

This algorithm guarantees that the obtained DOM tree is valid and at a predictable distance of the original tree if the original file was at distance  $k$  to the DTD. Notice that the number of \* nodes is a first estimate of the distance. It is  $O(n)$  if  $n$  is the size of the tree, but exponential in the distance  $k$ . We therefore consider Local Correctors where a local correction is made at each \* node, in order to remove this exponential constant factor.

#### 3.2 Local Correctors

A  $d$ -local Corrector makes corrections at each \* node, looking at a finite neighborhood at distance  $d$ , below the node when we proceed top-down. Clearly local corrections may not be as good as a global correction, but there are more efficient as they remove the exponential constant factor  $2^k$ . We propose a 1-local Corrector, or Local Corrector, i.e. look at the direct successors of a \* node or neighborhood of size 1 below the node.

A local correction at a \* node proceeds as follows. We propose a new label  $t$  for a \* node which guarantees that the

tree  $T$  will be valid. We consider the string  $s$  of labels of the successors of the \* node, select a rule of the DTD which minimizes the Edit distance and apply a local correction.

Local Corrector

**Input:** a file  $F$  and a DTD.

**Output:** a valid file  $F'$  close to  $F$ .

1. **Inductive \*-Marking of a tree,** as in the general corrector.

2. **Iterate Top-down at each \* node.** Consider all valid tags, i.e. tags which leads to a valid tree, for a \* node: choose the local correction which minimizes the Tree Edit distance in the local neighborhood.

Local Correction

**Input:** a \* node with a new label  $t$ , the string  $s$  of labels of successor nodes and a DTD.

**Output:** the closest string  $s'$  of the DTD, the Edit distance and a sequence of local corrections at the \* node.

1. Consider all rules of type  $t : m$  where  $m$  is a content model, i.e. a regular expression in the language of  $\mathcal{L}$ .

2. Compute the string Edit distance between  $m$  and  $s$ . Select the rule with the minimum distance and closest string  $s'$ . Make the local correction, i.e. the sequence of Deletion, Insertion and Modification to  $s$  to obtain  $s'$ .

The key function is to compute the distance between a string and a regular expression and there are efficient algorithms for this problem. We then obtain a distance and a corrected string  $s'$  where nodes are added, removed or modified. In order to solve very efficiently the distance to a regular expression problem, we assume the DTD is in Unary Normal Form, i.e. each rule  $t : m$  where  $m$  is a content model uses regular expressions where the \*, + operators are applied to single tags.

This algorithm is  $O(n)$  and always provides a valid file. Its drawback is that the corrected file  $F'$  may be far from  $F$ , i.e. we don't always obtain the best correction.

#### 3.3 Java Implementation Details

An implementation of the corrector using the notion of local correction at every error node is available on the site <http://www.lri.fr/~mdr/xml>. It uses the Sax and Xerces parsers and its key features are:

##### 3.3.1 Phase 1: Parsing

As we read the file, we scan the DTD with the Sax Parser, stores the declaration of each element in a class and obtain the DOM structure of the document by parsing it with the Xerces-j parser. We travel the DOM structure Bottom-up, and assign *Height* and *SubtreeSize* to each node as attribute. *Height* will be used to set the preference for correction whereas *SubtreeSize* will be used to calculate the approximate edit distance. We isolate the error nodes (\* nodes) along with the error statement as we consider several parsing possibilities. We store these nodes in an array. If there is any fatal error (i.e., the given XML document is not well formed) then exit, else go to the Correction phase.

### 3.3.2 Phase 2: Correction

For each \* node, we derive possible tags for that node such that the root of the tree accepts. For each possible tag  $t$ , we access the DNF of its Content Model, the minimum height of its subtrees (to compute the Edit distance while inserting or removing nodes) and the string  $s$  of tags of the successors. For each content model  $m$ , compute the Edit distance between  $s$  and  $m$ . Choose the model  $m$  with the minimum Edit distance. We obtain the string  $s'$  closest to  $s$  and a sequence of basic operations among M (Match), D (Delete), I (Insert), C (Change) which have to be realized for each letter of  $s$ . Modify the tree following these modifications. Check the attributes of each child node. If some attribute is missing then add it, if there is an undeclared attribute then remove it and if the attribute is not correctly defined then modify it.

Compute the total Edit distance, for all the corrections done. Remove the attributes added as part of this program. Serialize this DOM structure to an XML Document.

### 3.4 Example of a corrected file

Consider the following DTD which defines a class of right-branch trees.

```
<?xml version="1.0"?>
<!DOCTYPE a [<!ELEMENT a (l,r)><!ELEMENT r ((l,r)lq) >
<!ELEMENT l (#PCDATA) ><!ELEMENT q (#PCDATA) >]
```

Consider a file  $F$  whose DOM tree is represented in (a) of the following figure. At parsing time we generate two \* nodes. Correcting top-down, the first \* node is labelled  $r$  and the string  $s$  below at distance 1 is  $r$ . There are two possibilities for  $s'$ :  $s'_1 = l, r$  or  $s'_2 = q$  from the DTD. In the first case, we correct by introducing a left branch  $l$  and the Tree Edit Distance is 1. In the second case we drop the subtree labelled  $r$  and replaces it by a branch labelled with  $q$ . The Tree Edit Distance is the size of the subtree, i.e. 11 and this choice is not taken. We proceed until the next \* node labelled  $r$  and  $s = l, r, q$ . In this case  $s' = l, r$  and we drop the  $q$  branch. The total Edit Distance is 2.

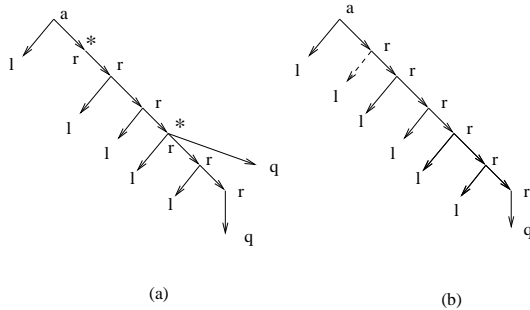


Figure 1: Correction of a DOM tree.

## 4. COMPARATIVE STUDY

We consider three different DTDs and analyze the parsing and the correction time of files up to 800 nodes. The first DTD defines deep binary trees, the second DTD defines wide

trees and the third DTD defines a mixture of deep and wide trees.

### 4.1 Deep Tree example

The first DTD defines the class of right-branch trees with the tags  $a, l, r, q$ , as in the example of section 3.5.

Given a valid binary tree, we randomly add extra branches to a node  $r$ , remove some left branches and change some of the  $r$  tags. Assume a distance of 10, i.e. we add 10 errors to various files from 50 nodes to 800 nodes.

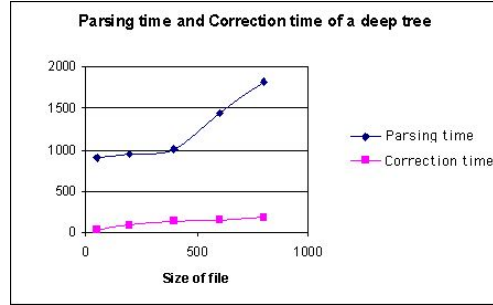


Figure 2: Analysis of Deep trees.

The Figure 4 shows the approximate time taken for the parsing as well as the correction of the XML document. It clearly indicates a correction time negligible compared to the parsing time. In this case, the possible expanded words  $w$  are very short and the Edit distance computations are very efficient.

### 4.2 Wide Tree Example

The second DTD is closer to classical examples in databases where the branching degree is large. We represent a document with a title and many lines, each one composed of many chars.

We introduce 10 random errors in wide valid trees of size 50 up to 800 nodes.

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE d [
<!ELEMENT d (title,line*)><!ELEMENT line (b,char*,c)>
<!ELEMENT title (#PCDATA)><!ELEMENT char (#PCDATA)>
<!ELEMENT b (#PCDATA)><!ELEMENT c (#PCDATA)>]
```

In this case, the discrepancy in the parsing and correction time is due to the large number of expansions of the regular expressions to be considered. This number increases drastically and we need to compute the edit distance for many probable corrections and then sort out the smallest one. Notice that the correction time is still less than the parsing time.

### 4.3 Deep Tree having some wide branches

We now define a mixture of the previous DTDs. We have many  $b$ s nodes below which we attach a right-branch tree, as defined by the first DTD. Starting from valid trees, we generate invalid trees with 10 random errors of size 50 up to 800 nodes.

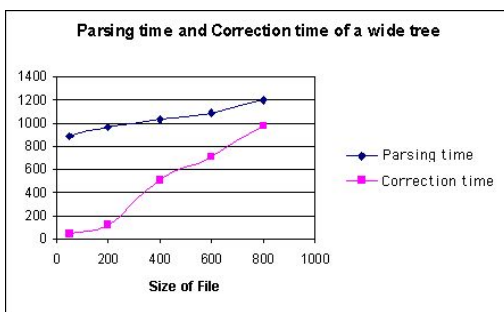


Figure 3: Analysis of wide trees.

```
<?xml version="1.0"?><!DOCTYPE b [<!ELEMENT b ((a|b),r)>
<!ELEMENT a (l,r)><!ELEMENT r ((l,r)|q ) >
<!ELEMENT l (#PCDATA) ><!ELEMENT q (#PCDATA) >]>
```

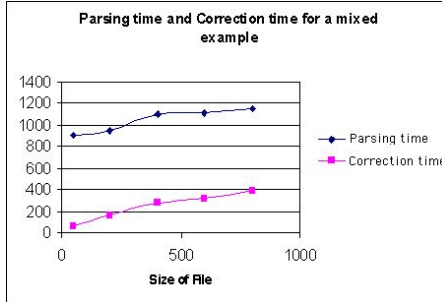


Figure 4: Analysis of deep and wide trees.

In this last case, the correction time is the average of the two previous example and significantly less than the parsing time.

If we had considered the global correction, the situation would have been quite different. We would need to check all the permutations of all possible corrections at each error node and then select the best correction. For deep trees, it would be very inefficient whereas for wide trees, it would be acceptable. The global method may have however produced a better correction for deep trees.

## 5. RANKING DOCUMENTS

Consider  $m$  distinct DTDs  $M_1, \dots, M_m$ . The rank of a document  $F$  relative to these DTDs is the vector whose values  $v_i = 10 - d_i(F)$  where  $d_i(F)$  is the estimated distance between  $F$  and  $M_i$ . This measure is interesting for search engines but needs to be generalized to close DTDs in order to handle multilingual documents.

The correction algorithm can be generalized to DTDs which may not be the original DTD of the file, provided they are close. If a document  $F$  is a french book with a DTD close to the english one, we wish to find a mapping between french and english tags and an approximate distance between the french document and the english DTD.

We generalize the distance between a tree and a DTD to a distance between two DTDs by defining a function of  $n$ , as

the minimum of the distances between all trees of size  $n$ . When they are close, i.e. the distance is  $O(1)$ , we propose to adapt the previous corrector.

### 5.1 Distance on DTDs

Consider two regular expressions or two DTDs. We first consider the case when they use the same language of tags  $\mathcal{L}_1$  and then different languages  $\mathcal{L}_1$  and  $\mathcal{L}_2$ .

DEFINITION 2. The distance between two regular expressions  $r$  and  $t$  on the same language  $\mathcal{L}_1$  is the function

$$Dist(n, r, t) = \text{Min}_{w \in r, w' \in t, |w|=|w'|=n} \{ dist(w, w') \}$$

The distance between two DTDs  $M_1$  and  $M_2$  on the same language  $\mathcal{L}_1$  is the function

$$Dist(n, M_1, M_2) = \text{Min}_{F \in M_1, F' \in M_2, |F|=|F'|=n} \{ dist(F, F') \}$$

We say that two DTDs are at distance  $O(1)$  or at constant distance if  $Dist(n, r, t) = O(1)$ . We are interested in short distances, say up to  $\log n$ , as we wish to capture close DTDs. In the sequel we only consider DTDs at constant distances.

Example. Let  $\mathcal{L}_1 = \{a, b, c, d, u, v\}$ . If  $r = au * bv * cd$  and  $t_1 = au * abv * c$  then  $Dist(n, r, t_1) = O(1)$ . If  $t_2 = ab * v * c$  then  $Dist(n, r, t_2) = O(n)$ .

If regular expressions or DTDs use different languages  $\mathcal{L}_1$  and  $\mathcal{L}_2$ , we wish to generalize this definition. We consider mappings  $\pi$  from  $\mathcal{L}_1 \cup \{\perp\}$  to  $\mathcal{L}_2 \cup \{\perp\}$  such that  $\pi(\perp) = \perp$ . We then extend the previous definition to two regular expressions  $r$  and  $t$  by taking the Minimum over all  $\pi$  of the distance between  $\pi(r)$  and  $t$ . We interpret  $\perp$  as the empty symbol, i.e.  $a.\perp = \perp.a = a$

DEFINITION 3. Two regular expressions  $r$  and  $t$  are isomorphic if there exists a mapping  $\pi$  between the symbols (tags) of  $r$  in  $\mathcal{L}_1$  and the symbols (tags) of  $t$  in  $\mathcal{L}_2$  such that  $\pi(r) = t$ . Two regular expressions  $r$  and  $t$  in Unary Normal Form such that  $|r| \geq |t|$  are homomorphic if there exists a mapping  $\pi$  between  $\mathcal{L}_1 \cup \{\perp\}$  and  $\mathcal{L}_2 \cup \{\perp\}$  such that the distance between  $\pi(r)$  and  $t$  is  $O(1)$ .

In case  $|r| \leq |t|$ , consider mappings from  $\mathcal{L}_2$  into  $\mathcal{L}_1$  and compare  $\pi(t)$  with  $r$ .

Example. Let  $\mathcal{L}_1 = \{a, b, c, d, u, v\}$  and  $\mathcal{L}_2 = \{a, b, c, d, u, v\}$ . If  $r = au * bv * cd$  and  $t_1 = cx * y * e$  then  $Dist(n, r, t_1) = O(1)$  with  $\pi$  such that  $\pi(a) = c, \pi(u) = x, \pi(b) = \perp, \pi(v) = y, \pi(c) = e$  and  $\pi(a) = c$ . Hence  $au * bv * cd$  and  $cx * y * e$  are homomorphic.

If  $t_2 = cx * y * e * e$  then  $Dist(n, r, t_2) = O(n)$  for any mapping  $\pi$  and these regular expressions are not homomorphic.

We generalize now the previous definition to two DTDs on different languages, assuming they are reduced, i.e. each rule influences the tree language associated with the DTD.

Consider that the DTD provides the *DNF* form for each tag. For a tag  $a$ , let  $DNF(a)$  be the regular expression which defines  $a$ . Suppose without loss of generality that  $|\mathcal{L}_1| \geq |\mathcal{L}_2|$ . Call a tag  $a$  *recursive* if either  $a^*$  occurs in some content model or if  $a$  occurs in the *DNF* form of  $a$  or if  $a$  is on a loop in the dependency graph whose nodes are tags and edges link a tag  $b$  with all the tags in its *DNF*.

**DEFINITION 4.** *Two DTDs  $M_1$  and  $M_2$  with roots  $r_1$  and  $r_2$  are homomorphic if there exists a mapping  $\pi$  between  $\mathcal{L}_1 \cup \{\perp\}$  and  $\mathcal{L}_2 \cup \{\perp\}$  such that 3 conditions hold: (1) if  $a$  is recursive in  $M_1$  then  $\pi(a)$  is recursive in  $M_2$  and  $\pi(DNF(a))$  is isomorphic to  $DNF(\pi(a))$ , (2) if  $b$  is non recursive in  $M_1$  then  $\pi(DNF(b))$  is homomorphic to  $DNF(\pi(b))$ , (3)  $\pi(r_1) = \pi(r_2)$ .*

These definitions lead to two simple observations:

**PROPOSITION 2.** *Two homomorphic regular expressions  $r, t$  are at distance  $O(1)$ . Two homomorphic DTDs are at distance  $O(1)$ .*

Given a file  $F$  with a DTD  $M_1$ , and another external DTD  $M_2$ , we wish to estimate the distance between  $F$  and  $M_2$ . We can look for all possible  $\pi$  which are homomorphisms between  $M_1$  and  $M_2$ , and look at the distance between  $\pi(F)$  and  $M_2$  with the previous corrector. This procedure would be exponential in the number of tags and inefficient in practice. We can however generalize the approach of a corrector as follows.

*Generalized corrector.* Consider the local corrector where we build a potential mapping  $\pi$ , bottom-up and estimate at each stage the Tree Edit distance. If it is greater than  $k$ , we stop and consider another mapping. If we reach the root with a potential mapping  $\pi$ , we correct  $\pi(F)$ .

We believe this heuristics may lead to efficient correctors on the average, as we indirectly solve the distance between two languages, a hard problem. It is an interesting open question whether or not we can test the approximate equivalence of two regular languages for the Tree Edit distance with moves.

## 6. CONCLUSION

A corrector for XML documents can estimate in linear time the distance between a document and a DTD when this distance is small. It is a useful operation to rank documents relative to several DTDs. The proposed implementation of a local corrector uses the Xerces parser and shows that in general the correction time is less than the parsing time and therefore scales up. It only corrects documents for a fixed constant distance but we intend to generalize the corrector for the Tree Edit distance with moves. In this case we would correct for a distance up to  $\epsilon \cdot n$ , as predicted by the theory.

We proposed the generalization of a corrector to external DTDs, i.e. to handle a file  $(F, M_1)$  whose DTD is  $M_1$  with an external DTD  $M_2$ , provided the distance between  $M_1$  and  $M_2$  is constant.

## 7. REFERENCES

- [1] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *IEEE Symposium on Foundations of Computer Science*, 1999.
- [2] A. Apostolico and Z. Galil. Pattern matching algorithms, chapter 14: Approximate tree pattern matching. *Oxford University Press*, 1997.
- [3] M. Blum and S. Kannan. Designing programs that test their work. *ACM Symposium on Theory of Computing*, pages 86–97, 1989.
- [4] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *ACM Symposium on Theory of Computing*, pages 73–83, 1990.
- [5] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD*, pages 493–504, 1996.
- [6] G. Cormode. Sequence distance embeddings. *Ph.D. thesis, University of Warwick*, 2003.
- [7] M. de Rougemont. A corrector for XML. *ISIP: Franco-Japanese Workshop on Information Search, Integration and Personalization, Hokkaido University*, <http://ca.meme.hokudai.ac.jp/project/fj2003/>, 2003.
- [8] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. In *IEEE Symposium on Foundations of Computer Science*, pages 339–348, 1996.
- [9] F. Magniez and M. Rougemont. Property testing of regular tree languages. Submitted for publication, <http://www.lri.fr/~magniez/PAPERS/mr04.pdf>, 2004.
- [10] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the fifth International Workshop on the Web and Databases*, pages 61–66, 2002.
- [11] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25:23–32, 1996.
- [12] K. C. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26:422–433, 1979.
- [13] Tidy. *HTML Tidy Library Project*, <http://tidy.sourceforge.net>. 2000.
- [14] R. Wagner and M. Fisher. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21:168–173, 1974.