CLASSIFICATION OF XML DOCUMENTS

M. HASSLER, A. BOUCHACHIA, R. MITTERMEIR

Department of Informatics Systems,
Alpen-Adria Universität Klagenfurt, Klagenfurt, Austria
E-mail: {hassler|hamid|roland}@isys.uni-klu.ac.at

Abstract

XML has progressively gained importance as a standard for formatting online documents. This development is perceived in many applications such as web services and digital libraries. It is therefore essential to design classification algorithms for XML documents in order to make information organization and search more effective.

In this paper, a classification approach for XML documents relying on instance-based learning ideas is proposed. As of primary interest, the suggested approach is dedicated to XML classification using the structure of documents. For the sake of completeness, extensions dealing with the documents' content are considered as well. The evaluation, using INEX's XML collections, confirmed the relevancy of the structure-based approach.

Keywords: XML, classification, document structure, document content, comparison, tree edit distance, instance-based learning

1 Introduction

The eXtensible Markup Language (XML) has recently emerged as a standard for developing many web applications dealing with document storage and retrieval, e.g., digital libraries. XML was mainly developed to achieve an enriched representation of documents and more retrieval flexibility when searching information. Systems concerned with such web applications are mainly augmented traditional information retrieval systems [1].

In contrast to traditional text categorization systems that deal with flat documents, XML classification systems involve the logical structure of documents

in addition to the raw text (pure content). Often such a structure is considered only as auxiliary information source useful for document representation. However, as will be shown later in this paper, it can be a primary source for document representation and a highly discriminating factor for a classification system.

Structurally, an XML document has a tree-like representation. A node in this tree (also known as *XML component*) is identified by a label, called tag, and some textual content. From an information search and retrieval perspective, the goal is to retrieve only relevant components instead of the whole documents in response to the user queries leading to retrieval precision improvement. On the other hand, from a classification perspective, the goal is to assign XML documents either document-wise or component-wise to predefined classes. The most frequently used scheme is the former one which is also the subject of the present work.

Classifying XML documents can basically be done in three ways:

(1) using exclusively the textual contents of XML documents as usually done in traditional text categorization systems,

(2) using exclusively the structure of XML documents, and

(3) using both, the contents, and the structure, in a hybrid manner.

This work is concerned with the latter approach. The aim is to discover structural and content patterns (characteristics) shared by XML documents of the same class. These patterns are mainly expressed in terms of their tags (node labels), contents, and inter-relationship.

Usually, classification is preceded by a training step. In the training phase, sets of XML documents belonging to each class are used to create representations of the classes. In the classification phase, new documents are compared with these representations in order to assign them to one or more classes. However for XML classification which involves the documents structure, the task of finding representations is highly difficult and expensive.

To cope with this complexity, an instance-based learning approach is proposed in this paper. This can support document retrieval by automatically labeling documents based on their relative similarity to known documents.

Instance-based learning comprises two main classification strategies: case-based learning and the k-nearest neighborhood algorithm (k-NN) [11]. The latter is more appropriate for the problem of XML classification. It departs from the assumption that similar documents should have the same class label. It uses a test document as a query and fetches the k training documents most

similar to the query. Majority voting is then used to determine the class of the query.

Classification in the context of information retrieval is closer to the problem of filtering [1], where there are two classes: relevant and non-relevant. This means that for a test document (i.e., query), the classification system predicts its corresponding topic (class = relevant). The other topics (class = non-relevant) are not of interest. Due to this relationship between information retrieval and classification, instance-based classifiers and in particular k-NN have been chosen. Quite typical and interesting such classifiers allow both, to perform classification, and to retrieve documents in a straightforward way. Other classifiers learn just the representation of classes, therefore no direct retrieval can be performed. k-NN, on the other hand, assumes that documents are entirely available and their classes are not encoded using some external representation (e.g., weights in neural networks).

Particularly important for the k-NN application is the distance which plays a central role in performing the classification. This importance becomes more crucial when the data objects to be classified are structured (i.e., objects are not merely a vector of values) as in the case of XML document classification where objects have the structure of trees. In this work, the distance is expressed in terms of a transformation cost that is required to equate two documents considering their structural and textual features. In other terms, such a distance measure needs to involve editing mechanisms, hence the choice of an *edit distance*. It corresponds to the minimum cost required to transform one document tree into another, taking content and structure into account. To the best of our knowledge, no previous work has used edit distance such that the content of an XML tree is also considered.

The rest of the paper is organized as follows. Section 2 reviews related work on XML document classification. Section3 explains the classification with two alternative approaches to compute the distance. In Sec. 3.1, the edit distance algorithm is discussed with respect to the content and structure before introducing a more content-dominated approach in Sec. 3.2. In Sec. 3.3 the k-NN algorithm is briefly explained. Section 4 discusses the experimental evaluation of the approaches before concluding.

2 Related Work

Although classification has been widely discussed in the framework of traditional text categorization where documents are flat, it has not yet attracted enough attention in the framework of structured documents. In the following,

some of the research work dedicated to XML document classification is briefly summarized.

Denoyer and Gallinari [9] proposed a classification approach that aims at using both, structure and content, to classify XML documents. It relies on a generative Bayesian classifier. An XML document is represented in the conventional tree-like form of a directed acyclic graph, where each node of the graph represents an XML component and each edge represents a parental relationship between the nodes. The generative model assumes that there are two types of belief, structural and textual, which are combined to get one single evidence about a document assignment to classes as suggested by the following formula:

$$P(doc|\theta) = P(stru|\theta) \times P(cont|stru, \theta)$$

where θ is a set of model parameters, *doc*, *stru*, and *cont* designate document, structure, and content respectively. The quantities $P(stru|\theta)$ and $P(cont|stru, \theta)$ are computed via a belief network as follows:

$$\begin{aligned} P(stru|\theta) &= \prod_{i=1}^{\#\ tags} P(node_i|ancestor(node_i)) \\ P(cont|stru, \theta) &= \prod_{i=1}^{\#\ terms} P(term_i|node_j, \theta) \end{aligned}$$

where the former considers the parental relationship between nodes and the latter the content (index terms) of nodes. During the training phase, the parameters of the generative model are learned using the Expectation-Maximization method. Unseen XML documents are assigned to the class with the highest probability $P(doc|class_k)$.

Bratko and Filipič [3] describe a similar approach to that described in [9]. Their method, however, is less general, since it assumes that all documents have the same structure across all classes. According to this method, documents are broken down into components, each of which contains either structured data or non-structured textual data. Let d_j, c_k, s_l, and t_i designate respectively a document j, a class k, a component l, and a term i, the total probability of a document is the product of the individual component probabilities:

$$P(d_j|c_k) = \prod_{s_l \in d_j} \prod_{t_i \in s_l} P_{s_l}(t_i|c_k)^{f^j(t_{is})}$$

where $f^j(t_{is})$ indicates the word frequency of term i in the s^{th} component of document j. The first product is over all structural components s_l that are

present in the document d_j. The probability P_{s_l} is obtained for each document component. From this, it is clear that each document is represented by a set of feature vectors, one for each component, so that word frequency $f^j(t_{is})$ is maintained on a per-component basis.

Zaki and Aggarwal [24] developed a classifier they called XRules. XRules is structure-oriented and aims at discovering a set of structural rules that define the individual classes. Basically, these rules, reflecting regular structural patterns of each class, are learned during the training phase. It is based on the assumption that the presence of a particular structural pattern in an XML document is related to the likelihood of its belonging to a particular class. During classification, for a given new document, the relevant rules are identified and certain statistics from all matching rules are combined to predict the most likely class for that document.

The other aspect which is important for the present work is tree edit distance. Work on classification of structured data frequently relies on edit distance to measure the dissimilarity between graphs/trees [17]. An edit distance algorithm computes the minimum cost of transforming one tree into another, where each elementary transformation (e.g., insert, delete, alter) is associated with a certain cost. In the domain of XML classification, usually XML peculiarities such as tags, parent-child relationships, root-leaf XPaths, arity of nodes, etc. [26, 4] are used as descriptive features. Different algorithms have been proposed to compute the edit distance between such trees [7, 8, 25, 20, 16]. Most of them use dynamic programming techniques as initially described in [14] in order to find the cheapest sequence of transformations, called *delta script* or *edit script* [7] which can be formulated as:

$$\begin{aligned} dist(T_1, T_2) &= \min_j \{s_j\} \\ s_j &= \sum_i^{L_j} t_i^{(j)} \end{aligned} \tag{1}$$

where s_j is an overall cost of a script j (of length L_j), and $t_i^{(j)}$ is the cost of a transformation i, part of the script j.

The main difference among the various edit distance measures rests in the set of edit operations allowed and their associated costs. Early work [19] considered *insert* and *delete* of *leaf* nodes, and node *relabelling* of *any* node. Extensions were then proposed to allow insertion and deletion of nodes anywhere in the tree [21, 22, 20].

In general the problem of finding the edit distance between two trees is NP-hard [15, 2]. The work described in [25] and [7] intends to find an efficient

algorithm for the reduced problem of ordered/binary trees, in which a left-to-right order among siblings is applied. Chawathe et al. [7] first applied the same edit operations (delete, insert, relabel) and restrictions as used by Selkow [19] to detect differences in structured documents. In subsequent works this approach is extended to cover also *move* operations [6]. In [5] operations for *copying* and *gluing* of subtrees were added.

Zhang and Shasha [25] provide a fast algorithm to calculate the edit distance between ordered labelled trees. The minimum costs of mapping all descendants of a node is computed in advance, using the notion of *keyroots*. Keyroots of a tree are defined as the set of all first-level children having left siblings plus the root node itself. Computing the keyroots of a tree in advance applies the concepts of *tree distance* and *forest distance*. The tree distance is calculated as the distance between two trees without considering the context of ancestors and/or siblings. The forest distance instead uses the tree distance plus takes ancestor and sibling relations into account. To get the minimum transformation cost of two trees, the minimum cost mapping from all keyroots amongst the children and the cost of the leftmost child (forest distance of its rightmost child) is needed. The algorithm then proceeds from the leaf nodes up to the root node in a postorder traversal.

Further edit distance methods dedicated to XML documents can be found in the literature related to structural mapping, e.g., change detection [8]. A good overview of existing algorithms for change detection can be found in [17].

3 XML Classification

In k-NN, the most important phase is the deployment phase. Each new unknown document can be seen as a query which will be used to fetch the k most similar neighbors. Relying on (weighted) majority voting, the new document can be flagged with the label of the winning class. Clearly, similarity plays a central role in k-NN. As introduced in Sec. 1, the distance (or equivalently the similarity) used in this work is the tree edit distance (TED) due to the structure of XML documents. In the following, the technical formulation of TED is provided. First a structure-oriented TED is described before extending it to include document's content as well (content- and structure-oriented). A third, component-based variant (CM) is suggested. There, structure is considered only indirectly. Hence, more importance is given to content. Once an appropriate distance metric is fully defined, the application of k-NN becomes straightforward as will be shown hereafter.

3.1 Tree Matching via Edit Distance

The matching mechanism used dwells on the algorithm proposed by Nierman and Jagadish [16] which relies on dynamic programming to align a source tree with a target tree. Whilst this algorithm is originally structure-oriented, in this work it is further enriched and generalized to deal also with the documents' content. In a nutshell, the algorithm considers:

i. simplified edit operations, and

ii. the content attached to XML nodes

This results in a hybrid distance measure that operates on augmented trees with textual contents. Full descriptions are given in the following two subsections.

3.1.1 Structure Matching

To formulate the projected distance, some definitions of the basic concepts are introduced.

Definition 1 (Tree) *A tree is defined as $T = (N, E)$ where N is the set of nodes and E the set of edges that indicate the parental relationship between nodes. A node $n \in N$ is associated with a label $\lambda(n)$, a content $\gamma(n)$, a parent node $p \in N$, and a set of child nodes, $ch(n) \subseteq N$. If n has no content, $\gamma(n)$ is empty (null). The root node, $root(T) \in N$, has no parent node, a child node having no descendant ($ch(n) = \emptyset$) is called leaf. The child nodes of n are uniquely identified as $n^1, n^2, \ldots, n^k$, where k is the degree of n denoted as $deg(n)$. Further, $|T| = |N|$ represents the total number of nodes in the tree T.*

Definition 2 (Ordered Tree) *An ordered tree $T = (N, E)$ is defined as a rooted tree, where a left-to-right order among the child nodes is set. The subtrees of T are identified as the trees whose root nodes are the child nodes of $root(T)$, denoted as $T^1, T^2, \ldots, T^k$ ($k = deg(root(T))$).*

Based on these definitions, the following basic edit operations together with their corresponding costs are defined:

Definition 3 (Insert operation) *Given a tree $T = (N, E)$ and a node $p \in N$, a leaf node n can be inserted into T as the i^{th} child of the node $p \in N$ using the operation ins(T,n,p,i). The cost associated with this operation is $C_{ins}(T, n, p)$.*

Definition 4 (Delete operation) *Given a tree $T = (N, E)$ and a leaf node $n \in N$ placed as the i^{th} child of a node p, the node n can be removed using the operation del(T,p,i). The cost of deletion is $C_{del}(T, n, p)$.*

Definition 5 (Alter operation) *Given two nodes n_1 and n_2 with labels $\lambda(n_1)$ and $\lambda(n_2)$. The label of n_1 can be replaced with the label of n_2 using the operation $alt(n_1, n_2)$ ($\equiv \lambda(n_1) \leftarrow \lambda(n_2)$). The cost associated with this operation, $C_{alt}(n_1, n_2)$, is defined as:*

$$C_{alt}(n_1, n_2) = \begin{cases} 0 & \text{if } \lambda(n_1) = \lambda(n_2) \\ \beta\ (\in \mathbb{R}^+) & \text{otherwise} \end{cases} \tag{2}$$

These operations are associated with a fixed cost and provided as parameters to the routine for calculating the distance. Quite appealing, if $C_{ins}(T, n, p) = C_{del}(T, n, p)$, the edit distance measure becomes symmetric. Note that one can parameterize the costs of inserting, deleting, and altering. This seems reasonable for taking into account additional information such as a node's depth in the document tree or semantic closeness between tags (i.e., title and subtitle in papers).

The operations which are defined on leaf nodes can be generalized to operations on inner nodes, where an inner node is the root of a subtree. Consequently, the cost of applying an operation on an inner node is recursively computed by summing up the costs of manipulating its descendants. Hence, the following definitions of the cumulative costs are given:

Definition 6 (Recursive insert) *The insertion of a whole subtree S into a given tree T is done recursively using subtree insert (Def. 3).*

Definition 7 (Recursive delete) *The deletion of a whole subtree S of a given tree T is done recursively using subtree delete (Def. 4).*

Definition 8 (Subtree insert/delete) *The cumulative cost of inserting (resp. deleting) a subtree S at node p from tree T is $C_{insCum}(T, S, p)$ (resp. $C_{delCum}(T, S, p)$) and computed as follows:*

$$\begin{cases} C_{insCum}(T, S, p) = C_{ins}(T, root(S), p) \\ \qquad\qquad + \sum C_{insCum}(T, S^j, root(S)) \\ C_{delCum}(T, S, p) = \sum C_{delCum}(T, S^j, root(S)) \\ \qquad\qquad + C_{del}(T, root(S), p) \end{cases} \tag{3}$$

In other words, at each node of the subtree S of the destination (resp. source) tree T_2 (resp. T_1), the cost of the recursive insert (resp. delete) operation is calculated by summing the cost of inserting (resp. deleting) the single node $root(S)$ with the cumulative cost of inserting (resp. deleting) each of its children S^j.

Having introduced some required variables, the edit distance between two trees T_1 and T_2 is computed using Alg. 1. For the sake of reference, let this method of computing the dissimilarity be called **TED_SO** as abbreviation for *tree edit distance using structure only*.

Based on dynamic programming, this algorithm constructs $distMat$, a $(deg(root(T_1)) + 1) \times (deg(root(T_2)) + 1)$ matrix of distance values between the nodes of the two trees.

The details of the algorithm are given as follows: First (line 4), the algorithm compares the root nodes of T_1 and T_2. The alter operation loads the value of β into cell $distMat[0][0]$ if relabeling is necessary. Otherwise, $distMat[0][0]$ will be initialized with "0". Then, as seen in lines 6 to 8 and 9 to 11, the algorithm computes the distance values of potentially inserting or deleting all nodes given the roots of two trees. These values serve to trigger the dynamic computation of cumulative adaptation costs, where each child of T_1 is recursively compared to each child of T_2. Thus, each cell $distMat[i][j]$ ($i > 0$ and $j > 0$) is assigned a cost that is computed in the following way, using the content of its neighboring three cells:

- the content of the upper left neighbor, $distMat[i-1][j-1]$, is added to the distance between the subtrees rooted at nodes n_i and n_j (i.e., T_1^i and T_2^j) (line 14). This case corresponds to a match between node n_i and node n_j.
- the content of the left neighbor, $distMat[i][j-1]$, is added to the cost of inserting a subtree T_2^j to the source tree (line 15). This case corresponds to an insertion of a subtree whose root is n_j.
- the content of the upper neighbor, $distMat[i-1][j]$, is added to the cost of removing a subtree T_1^i from the source tree (line 16). This case corresponds to a removal of an obsolete subtree whose root is n_i.

The minimum cost of these three alternatives is retained and stored in $distMat[i][j]$.

Fig. 1 depicts the process of cost computation when comparing two trees T_1 and T_2 (for completeness purpose content is also shown). Gray blocks indicate the nodes which are involved. Initially, the distance matrix contains the cost of altering the root nodes (if needed). The first row (resp. column) is filled by the cumulative costs of inserting (resp. deleting) every single node. All remaining cells are then computed as the minimum sum of previous costs and that of the current transformation represented as arrows in the figure. In order to match the inner nodes D in both trees, a further recursive call is carried

Algorithm 1 Tree Edit Distance algorithm

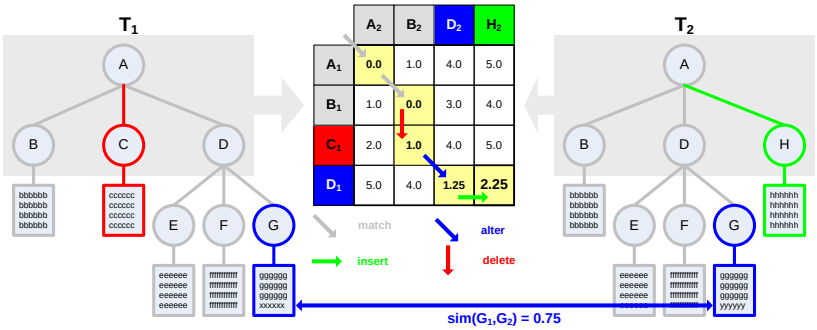
```
1:  procedure TreeDist(T1, T2)
2:      int M = deg(root(T1))
3:      int N = deg(root(T2))
4:      int[][] distMat = new int[0..M][0..N]
5:      distMat[0][0] = c_alt(root(T1), root(T2))
6:      for j = 1 to N do
7:          distMat[0][j] = distMat[0][j-1] + C_insCum(T2, T2^j, root(T2))
8:      end for
9:      for i = 1 to M do
10:         distMat[i][0] = distMat[i-1][0] + C_delCum(T1, T1^i, root(T1))
11:     end for
12:     for i = 1 to M do
13:         for j = 1 to N do
14:             distMat[i][j] = min{
15:             distMat[i-1][j-1] + dist(T1^i, T2^j),
16:             distMat[i][j-1] + C_insCum(T2, T2^j, root(T2)),
17:             distMat[i-1][j] + C_delCum(T1, T1^i, root(T1))
18:             }
19:         end for
20:     end for
21:     return distMat[M][N]
22: end procedure
```

out. The process terminates with the overall result (along a path) located in the bottom right corner.

The original algorithm proposed by Nierman and Jagadish [16] outputs only the edit distance between two trees. There is no way to reconstruct the optimal sequence of edit operations that led to the obtained final edit distance. To overcome that, all possible edit scripts that correspond to the minimal distance between the two trees at hand have to be memorized. In this context an edit script is defined as follows:

Definition 9 (Edit Script) *An edit script δ is an ordered sequence of edit operations that transform T_1 into T_2. In general, there exist an infinite number $\delta^1, \delta^2, \ldots, \delta^m$ of edit scripts that correctly transform T_1 into T_2.*

Definition 10 (Minimal Edit Script) *Let $\delta^1, \delta^2, \ldots, \delta^m$ be a set of correct edit*



$$\mathbf{TED(T_1,T_2) = 1 + 1 + (1\text{-}0.75) = 2.25}$$

Figure 1: Tree edit distance algorithm matrix (assuming $C_{ins} = C_{del} = 1$ for any leaf node)

scripts transforming T_1 in T_2. A minimal edit script is then defined as:

$$\begin{aligned} Min_{k=1..m}\{\delta^k\} = \delta^i \iff \forall \delta^j \mid i \neq j, \\ dist(T_1, T_2)^{\delta^i} \leq dist(T_1, T_2)^{\delta^j} \end{aligned} \tag{4}$$

where $dist(T_1, T_2)^{\delta^j}$ indicates the distance between T_1 and T_2 obtained after applying the script δ^j. Note that there might be more than one minimal edit script for a pair of trees.

As already proposed by Barnard et al.[2] the algorithm of Nierman and Jagadish can be extended to keep track of the minimal edit scripts. Instead of having *distMat* storing only cumulated editing costs, an additional entry in the matrix keeps track of the corresponding minimal edit scripts. After the algorithm finishes, $distMat[deg(T_1)][deg(T_2)]$ contains the minimal edit distance and the set of minimal edit scripts.

3.1.2 Content and Structure Matching

The second type of distance required to complete the computation of tree edit distance is the content-based distance. It is referred to as **TED_CAS** representing *tree edit distance using content and structure*. To define it, we still rely on Alg. 1. However the previously used cost functions have to be redefined in such a way that instead of fixed costs for node-manipulations the degree of similarity between the contents attached to the nodes compared is also taken into account.

With different types of content, different similarity functions seem to be appropriate. For textual content, various metrics like the standard Boolean or Vector Space model could be used [1]. In order to compare more complex contents like tabular, graphical, or pictorial information, other metrics might be more suitable [1]. Hence, the content matching algorithm is not specific about the similarity function to be used. Any properly defined content matching function could be substituted for $sim(\_, \_)$.

Let $sim(\gamma(n_1), \gamma(n_2))$ be a similarity function suitable for comparing the contents of two nodes n_1 and n_2 such that this similarity measure is normalized so that it takes values in the unit interval [0,1] (where $sim = 0$ means no match, $sim = 1$ indicates full match, and $sim \in \ ]0, 1[$ means partial match). The boundary conditions are defined as:

$$sim(null, null) = 1 \tag{5}$$

$$sim(null, \gamma(n_2)) = sim(\gamma(n_1), null) = 0 \tag{6}$$

Eq. 5 stipulates that the content-based similarity of two nodes with empty content is total (i.e., complete match), while Eq. 6 stipulates that content-based similarity between a node with some non-empty content and a node with empty content is 0.

The cost functions for *inserting* and *deleting nodes with their contents* are defined as:

$$C_{insCon}(n) = 1 - sim(null, \gamma(n)) = 1 - 0 = 1 \tag{7}$$

$$C_{delCon}(n) = 1 - sim(\gamma(n), null) = 1 - 0 = 1 \tag{8}$$

The total cost of altering the content, C_{altCon}, (Eq. 9) is the sum of the cost of changing the label $C_{alt}(n_1, n_2)$ (Eq. 2) and the cost of altering the content which is expressed as:

$$C_{altCon}(n_1, n_2) = C_{alt}(n_1, n_2) + \rho * (1 - sim(\gamma(n_1), \gamma(n_2))) \tag{9}$$

The amount ρ is a cost factor that scales up the dissimilarity of contents (since $sim(\gamma(n_1), \gamma(n_2)) \in [0, 1]$ and $c_{alt}(n_1, n_2)$ can be larger than 1).

Furthermore, we can decide whether the alter operation is cheaper than the combination of a delete and insert operation. This can be tuned by a user-specified parameter, α ($0 \leq \alpha < 1$). Precisely, if $sim(\gamma(n_1), \gamma(n_2)) > \alpha$ then $C_{altCon} < (C_{delCon} + C_{insCon})$ (alter is cheaper) and if $sim(\gamma(n_1), \gamma(n_2)) < \alpha$ then $C_{altCon} > (C_{delCon} + C_{insCon})$ (alter is more expensive); otherwise, alter has the

same cost as the sum of a delete operation and an ensuing insert. From these constraints, the cost factor ρ can be determined as follows:

$$\begin{aligned} \rho * (1 - \alpha) &= C_{delCon}(n_1) + C_{insCon}(n_2) \\ \rho &= \frac{C_{delCon}(n_1) + C_{insCon}(n_2)}{(1 - \alpha)} \end{aligned} \tag{10}$$

Based on these definitions, the following parameter changes have to be made in Alg. 1: Taking the similarity of content into account $C_{alt}(root(T_1), root(T_2))$ in Alg. 1 (line 4) is substituted for $C_{altCon}(root(T_1), root(T_2))$. Further, the subtree insert (resp. delete) operations are redefined to call C_{insCon} (resp. C_{delCon}) instead of C_{ins} (resp. C_{del}). The remainder of the algorithm stays unchanged.

3.2 Component-based Matching

While the focus of TED_CAS just presented has been on explicit consideration of the structure to compare documents, here an alternative, more content-oriented matching procedure is proposed. It uses the structure of documents only implicitly by exploiting the component-based view of XML documents. This method is referred to as **CM** standing for *component-based matching*.

Specifically, the idea is to measure the similarity between documents using only nodes with contents. Indeed, every node in the source tree is compared to all nodes in the destination tree relying on a content similarity matrix (denoted as *Content Matrix*). As this matrix obtains its information only on the basis of content containing nodes (mostly leaf nodes) of XML trees, the structural depth of the trees involved is ignored. However, as the comparison will be on a node per node basis, the fact that XML documents are not flat but structured and therefore meaningfully arranged containers of information is still considered.

A cell $contMat[i][j]$ refers to the degree of similarity between the contents of the node n_i in the source tree and the node n_j in the destination tree, i.e., $contMat[i][j] = sim(\gamma(n_i), \gamma(n_j))$.

One can compute content similarity by either comparing all content containers or define a two-step process for similarity comparison. In the latter the labels of the content bearing nodes serve as filter. If the labels of nodes to be compared are the same, then the contents of these nodes are compared. Otherwise, similarity is postulated to be 0. The user can determine via a flag variable, ζ, whether this coarser similarity match should be used. If ζ is set to *true*, the similarity of nodes with non-equal labels is set to 0.

Once the content similarity matrix, $contMat$, is filled, the distance between the corresponding documents can be computed using an algorithm that traverses that matrix in a single pass. Basically, three edit operations: insert, delete, and alter are applied. During the computation, these operations are labelled either as $safe$ (certain) or as $unsafe$ (uncertain) as shown in Alg. 2. The semantics of this label is that $safe$ operations can be performed right away while in the $unsafe$ case, application of the operation can involve further treatments (i.e., weighting strategies).

Basically the algorithm proceeds mainly in two steps: (1) computing the similarities, and (2) marking of nodes according to the edit operations as follows (item numbers correspond to lines in Alg. 2):

14: If $contMat[i][j] = 1$, then a node $n_{1,i}$ and the first node $n_{2,j}$ (minimal index j) are both marked as $safe\_match$ with no additional cost.

15: A source node, $n_{1,i}$, having no matching destination nodes ($contMat[i][j] = 0, \forall j = 1 \ldots |dest|$) is marked as $safe\_delete$. The cumulative cost is increased by the weighted cost of $safe\_delete$.

16: A destination node, $n_{2,j}$, with no corresponding source nodes ($contMat[i][j] = 0, \forall i = 1 \ldots |source|$) is marked as $safe\_insert$. The cumulative cost is increased by the weighted cost of $safe\_insert$.

17: An unmarked source node, $n_{1,i}$, is marked as $unsafe\_match$ along with one unmarked destination node $n_{2,j}$, if $n_{2,j}$ is the first node (minimal index j) fulfilling the condition $contMat[i][j] \geq \alpha$, where α is a user-specified similarity threshold.

18: Any remaining unmarked source node, $n_{1,i}$, ($contMat[i][j] < \alpha, \forall j = 1 \ldots |dest|$) is marked as $unsafe\_delete$.

19: Any remaining unmarked destination node, $n_{2,j}$, ($contMat[i][j] < \alpha, \forall i = 1 \ldots |source|$) is marked as $unsafe\_insert$.

As in the tree edit distance approach, each of the edit operations is associated with a certain cost. In addition, the labels $safe$ and $unsafe$ can be attached weights. Since the distance is inversely proportional to similarity, the transformation costs are calculated using the same formula applied in the tree edit distance approach taking $dist(n_{1,i}, n_{2,j}) = 1 - contMat[i][j]$ into account. The final distance between the documents is the sum of all (weighted) operation costs on account of the type of marking. As there is no recursive computation, the computation of the edit script is done in linear time and space.

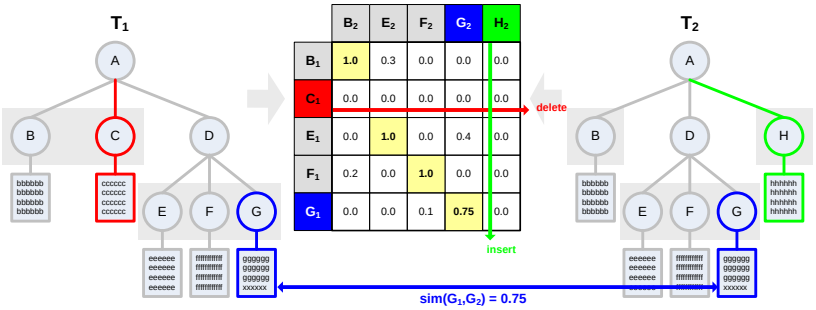
Algorithm 2 Component-based Matching algorithm

```
1:  procedure ContentDist(T_1, T_2, ζ)
2:      int M = |T_1| /*only content nodes*/
3:      int N = |T_2| /*only content nodes*/
4:      float[][] contMat = new float[0..M][0..N]
5:      for i = 1 to M do
6:          for j = 1 to N do
7:              if ζ = true and λ(n_1) ≠ λ(n_2) then
8:                  contMat[i][j] = 0
9:              else
10:                 contMat[i][j] = sim(γ(n_{1,i}), γ(n_{2,j}))
11:             end if
12:         end for
13:     end for
14:
15:     mark safe_match
16:     mark safe_delete
17:     mark safe_insert
18:     mark unsafe_match
19:     mark unsafe_delete
20:     mark unsafe_insert
21:
22:     return ∑ weighted_costs (based on marks)
23: end procedure
```

Fig. 2 depicts the process of cost computation when comparing two trees T_1 and T_2. Again, gray blocks indicate the nodes which are involved. Initially, the content matrix is filled with the pairwise similarities of all content nodes. Then, all $safe$ transformations are marked, including complete matches (nodes B, E, and F), inserts (node H) and deletes (node C). Nodes above a given similarity threshold (node G in both trees) are marked as $unsafe$ match (interpreted as an alter operation). The remaining unmarked nodes of T_1 (resp. T_2) are marked as $unsafe$ deletes (resp. inserts).

It is important to note that one can derive various variants of the CM-distance:

1. CM_S: where the tree nodes are compared only if they have the same tag ($\zeta = true$, see Alg. 2)
2. CM_A: where tags are ignored ($\zeta = false$)



CM(T_1,T_2) = 1 + 1 + (1-0.75) = 2.25

Figure 2: Component-based matching algorithm matrix (assuming $C_{ins} = C_{del} = 1$ for any node)

3. TED_CM: corresponding to a combination of TED_SO (Sec. 3.1.1) and CM_A. Here, the final distance of two documents is computed as follows: $\eta \cdot TED\_SO + (1 - \eta) \cdot CO\_any$, where η=0.5.

3.3 Overview of *k*-NN

The *k*-NN algorithm [11] is based on the assumption that the classification attributed to a sample should be most similar to the classification of other samples that are nearby in the space.

Compared to other learning methods such as probabilistic classifiers, *k*-NN does not rely on prior probabilities. Nevertheless, it is known for its effectiveness [23], though it does have efficiency problems. The main computational task involves sorting the training samples in order to find the *k*-nearest neighbors of a given query. The metric to be used for computing the distance is at the experimenters discretion. In case of XML documents, one of the proposed alternatives for computing the distance between documents (edit distance or component-based matching) seems promising. The remaining steps of attributing the most suitable class are straightforward as outlined below.

To use *k*-NN a collection of N training documents $X^T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ is required, where in the context of this work, x_i are XML documents and y_i are associated class labels. The elements of this set are used as reference samples for the *k*-NN algorithm. To assign a label to an unlabelled document (query), *doc*, the algorithm finds those k documents in the reference samples (labelled documents) that are the closest to it. The label shared by the majority of these k nearest neighbors is assigned to the query. As *k*-NN

Algorithm 3 Classification via the k-NN algorithm

Let X^L be the set of labeled documents, C the number of classes such that $\bigcup_j X_j = X^L,\ j = 1...C$
Let X^U be the set of unlabeled documents

procedure k-NN(k)
  for i = 1 to $|X^U|$ **do** // x_i is an unlabeled document
    for j = 1 to $|X^L|$ **do** // y_j is a labeled document
      $dist[j] = sim(x_i, y_j)$
    end for
    Sort $dist$[] ascending
    Select the first k documents ($\in X^L$ with smallest distance)
    Assign x_i to the class of the majority among the k documents. Break ties if more than one winning class exists.
  end for
end procedure

depends on the value of k, an appropriate choice of k has to be taken.

Note that k-NN is a lazy learning algorithm because no model needs to be built a priori. Steps of the classification procedure via k-NN are summarized in Alg. 3. Once the unlabelled XML document set is assigned class labels, the classification accuracy can be computed. It measures how often the algorithm's labeling decision meets the actual labels of the documents.

4 Evaluation

This section addresses the following aspects: How do different values of k influence the classification? What is the impact of training size on classification performance? How does content and structure matching perform compared to matching based on structure only? All of these questions are discussed using some real-world XML collections (MovieDB -movie database-) which were proposed in INEX'05 [12]. Descriptions of the collections can be found in [10]. Documents of these collections are assigned to 11 classes. Basically, the XML collections are of two types:

- Structure-only (SO) collections: These contain only the structure of the XML documents and include 4 collections: *m-db-s-0, m-db-s-1, m-db-s-2, m-db-s-3*, where the last 3 collections are noisy versions of the first one. The amount of noise increases from the first to the last collection. The collections originally come in the form of two sets (training and

testing sets) as follows: *m-db-s-0* (4824, 4816), *m-db-s-1* (4818, 4814), *m-db-s-2* (4820, 4809), and *m-db-s-3* (4821, 4809).

- Content-and-structure (CAS) collection: This collection is called *m-db-cs-1* and consists of 2415 training and 2410 testing documents. Both training and testing sets are reasonably large and therefore sufficient to adopt the two standard evaluation stages, training and testing, separately.

To answer the questions formulated earlier, 3 sets of experiments are run. The first two deal with the structure-only setting, while the last one is concerned with the content-and-structure setting. A comparison of our results against some available results from other authors is shortly highlighted at the end of this section.

In all experiments, the classification accuracy which measures how often the classifier's decision meets the actual assignment is used as measure of success. Formally it is defined as:

$$\text{Accuracy} = \frac{\text{\# correctly classified testing docs}}{\text{\# testing docs}} \tag{11}$$

4.1 Experiment I : How Does k Affect the Accuracy?

As explained in Sec. 3.3, the size of the neighborhood (k) is a key parameter in the k-NN algorithm. Therefore, one aspect to look at is to check the effect of k on the accuracy. For this purpose, the values 3, 5, 7, 9, 15, and 21 were used in the experiments. Note that only the SO collections are used and, due to time constraints, only a proportion (10%) of them is selected randomly and uniformly distributed over the 11 classes to show the effect of k.

Table 1: Parameter settings for TED

α (Eq. 10)	C_{ins} cost (Eq. 3)	C_{del} cost (Eq. 3)	β (Eq. 2)
0.5	1.0	1.0	2.0

Using Alg. 1 to run k-NN, and setting the required parameters as given in Tab. 1 ($\beta = C_{ins} + C_{del}$ to avoid permanent node relabelling), the results displayed in Tab. 2 are obtained.

The outcome of the experiments lead to the following conclusions: **(i)** as k increases, the accuracy of the algorithm monotonically decreases independently of the collection used, **(ii)** the noise introduced in the m-db-s-1/2/3 has

Table 2: Effect of k on the accuracy

Corpus	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$
m-db-s-0	0.922	0.920	0.918	0.903	0.892	0.889
m-db-s-1	0.903	0.892	0.891	0.897	0.885	0.866
m-db-s-2	0.874	0.868	0.858	0.849	0.802	0.790
m-db-s-3	0.862	0.868	0.860	0.852	0.825	0.814

negatively impacted the accuracy (as the amount of noise in relation to m-db-s-0 increases, the accuracy decreases), and **(iii)** the maximum drop in the accuracy is only 3.3% when raising k. Therefore, we will continue using the different values of k in the remaining experiments since these results do not allow to convincingly consider a particular k-value better than the others.

More interesting, the classifier provides very high accuracy results, but this remains relative to the amount of documents used in this experiment.

4.2 Experiment II - How Does the Training Data Affect the Accuracy?

Furthermore, k-NN uses the entire set of training samples as a basis to label the query. Hence, it is clear that the magnitude of the training set is crucial for the accuracy of the algorithm. To observe the effect of the size of the training data set, 5 subsets are derived from the *m-db-s-0* collection. They correspond to 10%, 30%, 50%, 70%, 100% of the available training data set. The proper subsets were constructed by random and uniform selection from the whole training data set ensuring that every subset contains all available labels.

Table 3: Effect of the training data on the accuracy

Size	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$
10%	0.922	0.920	0.918	0.903	0.892	0.889
30%	0.928	0.925	0.934	0.932	0.930	0.930
50%	0.924	0.929	0.928	0.925	0.925	0.923
70%	0.934	0.933	0.932	0.930	0.930	0.932
100%	0.934	0.934	0.932	0.932	0.929	0.931

Using the same setting described in Sec. 4.1, the results shown in Tab. 3 were obtained. Unexpectedly, the size of the training set did not greatly impact the accuracy of the classifier. The reason might lie in the inter-document similarity, meaning that the classes are highly homogeneous. Furthermore, the accuracy remains in the same range of values (regardless the value of k) without noticeable fluctuations when increasing the size of the training data.

4.3 Experiment III - How Does CAS Setting Affect the Accuracy?

To check the effectiveness of the proposed approach taking both, the content and the structure of XML documents into account, five methods are applied on the CAS collection (*m-db-cs-1*) described earlier. These are briefly shown in the following:

Method	Description
BM	A Boolean model [1] is applied as in traditional information retrieval where documents are represented as a bag of words. Specifically, a document is a vector of binary values; such that a value is 0 if the corresponding term is absent and 1 if the corresponding term is present in the document. The similarity between two document can be measured using the Jaccard coefficient which expresses the degree of equality (size of common terms divided by the union of exiting terms in both documents). Clearly, according to this method, the structure of documents is neglected.
TED_SO	Sec.3.1.1, parameters are set as in Sec. 4.1
TED_CAS	Sec.3.1.1, parameters are set as in Sec. 4.1
CM_S	Sec.3.2
CM_A	Sec.3.2
TED_CM	Sec.3.2

These methods are run on 20% of *m-db-cs-1* to obtain the results displayed in Tab. 4. It is worth noting that except in the TED_SO variant, the Jaccard coefficient [1] is used to compare the content of two components.

Although *m-db-cs-1* and *m-db-s-0* are different form each other, the accuracy of TED_SO on both collections (when using the structure-only) is nearly the same and exceeds 90% accuracy. More surprising is the fact that

Table 4: Effect of CAS on the accuracy

Method	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$
BM	0.327	0.352	0.352	0.331	0.335	0.313
TED_SO	0.916	0.907	0.895	0.895	0.856	0.860
TED_CAS	0.652	0.634	0.640	0.673	0.584	0.558
CM_S	0.352	0.360	0.305	0.309	0.296	0.272
CM_A	0.130	0.163	0.175	0.160	0.134	0.140
TED_CM	0.909	0.907	0.897	0.893	0.862	0.860

BM, CM_S, and CM_A perform so markedly worse. Furthermore, comparing the edit distance methods *TED_CAS* and *TED_SO*, it is worth concluding that including content deteriorates the accuracy. Indeed the difference in the accuracy is significant: 26%, 27%, 25%, 22%, 23%, and 31%. This is also true when entirely ignoring the structure, as with the BM method or when using the component-based matching described in Alg. 2 and relying on CM_S and CM_A. The accuracy deterioration in this case is much worse. More consistent with our expectations, combining TED_SO and CM_A, which results in TED_CM, allows to obtain much better results since this combination enables to consider the content while assuring high classification accuracy of the TED_SO method. The accuracy in this case remains high.

From these preliminary experiments, one can see that structure is a central aspect in the overall similarity between XML documents during classification. The fact that content did not contribute in improving the classification accuracy is counter-intuitive. However the results obtained are consistent with those obtained by XRules explored in [24] already mentioned in Sec. 2. XRules, which is entirely structure-oriented, has been evaluated on a real data set (constructed from Log reports) and a synthetic data set (constructed by a data generation program simulating website browsing behaviour). This classifier has then been compared against a traditional purely content-oriented vector space classifier (IRC) and a classifier based on associations (CBA). The empirical evaluation has shown that XRules outperforms both classifiers. Its accuracy is 2-4 % better when evaluated on the real data set and about 20% better on the synthetic data set.

A further reason for having TED_SO better than TED_CAS might be due the fact that the content in the MovieDB collections is relatively poor and not discriminative. Therefore, to further validate this result, additional experi-

ments on other document collections are certainly needed. However, at this stage, our work only relies on the MovieDB collections available at our hand. As long as we are concerned with the accuracy of the proposed approach, we need to conduct comparative studies against other results from the literature. Unfortunately, the only ones found are related to MovieDB; hence our motivation for using this collection.

4.4 Comparison

Because we provided a range of methods, it is relevant to check how these methods compare to the state-of-the-art methods that have been applied on the same collections. To do that, two references appearing in the INEX 2005 workshop are considered. The first is by Hagenbuchner et al. [13] who applied contextual self-organizing maps for structured data (CSOM-SD) in order to classify XML documents. Actually, CSOM-SD is dedicated to clustering rather than to classification. However in [13] they have been tested for classification purposes, using a measure called "classification performance". The second is by Candillier et al. in [4] who applied inductive decision trees (IDT).

To compare these methods against those proposed in this paper, one has to use the same evaluation metrics, accuracy, recall, and precision (at the macro and micro levels). Note that macro-averaging computes the recall and precision values for each class separately which are then averaged over all classes. Clearly, the contribution of all classes is the same (each class has the same weight). On the other hand, micro-averaging computes recall and precision for all documents without distinguishing between classes. Here, each document has the same weight or impact on the overall measurement [18].

Moreover, it is important to bear in mind that for the sake of this comparison, only the best performance rates achieved by each method are used (since with the same approach, several variations have been proposed and tested).

The results of the comparison is shown in Tab. 5. These illustrate that TED_CM largely outperforms CSOM-SD in terms of accuracy by a rate difference of 6%. However, when considering micro and macro recall, the IDT approach performs better than TED_CM. Unfortunately, recall without precision is not much telling. The precision values achieved by TED_CM are very encouraging especially when taking recall values into account.

Table 5: Classification Comparison for *m-db-s-0*

Approach	Accuracy	Micro Recall	Macro Recall	Micro Precision	Macro Precision
CSOM-SD	0.873	-	-	-	-
IDT	-	0.968	0.960	-	-
TED_CM	0.934	0.934	0.934	0.937	0.911

'-': means value not available

5 Conclusions

This work introduces a classification approach for XML documents based on the *k*-nearest neighborhood algorithm. For measuring the distance between artefacts, cost functions of various tree editing operations have been proposed. The originality of the approach comes from the fact that these distance metrics considers both, the content and structure, of XML trees. An initial evaluation indicates that this approach is very promising in the variations: 'structure-only' and 'content-and-structure'. The results also indicate that with XML documents structure bears more significance than the content does. This, on first glance surprising result is in accordance with some similar work reported in the literature. However the combination of edit distance and component-based matching offers the possibility to tune the weight of both, content and structure.

Although *k*-NN requires much time, its application is motivated by the possibility to use it for document organization via the classification and information retrieval as well. Its performance can be improved by reducing the number of comparisons. A systematic way to do this is by clustering labelled XML documents before applying *k*-NN. This will allow also to deal with noise. Further aspects can also be improved; for instance, instead of pure counts of classes (in majority voting), one can use weighted similarity as a score. From the XML perspective, it is still worth looking at the way the content is represented and the way it is combined with structure to form a reflective representation of XML documents.

References

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, ACM Press, 1999.

[2] D. Barnard, G. Clarke, and N. Duncan. Tree-to-tree correction for document trees. Technical Report 95-372, Department of Computing and Information Science, Queen's University, 1995.

[3] A. Bratko and B. Filipič. Exploiting structural information in semi-structured document classification. In *Proc. 13th Intl' Electrotechnical and CS Conf. (ERK)*, 2004.

[4] Laurent Candillier, Isabelle Tellier, and Fabien Torre. Transforming XML trees for efficient classification and clustering. In Fuhr et al. [12], pages 487–480.

[5] S. Chawathe. Comparing hierarchical data in external memory. In *Proc. 25th Intl' Conf. on Very Large Data Bases (VLDB)*, pages 90–101, 1999.

[6] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *ACM Intl' Conf. on the Management of Data (SIGMOD)*, pages 26–37, 1997.

[7] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proc. ACM SIGMOD Intl' Conf. on Management of Data*, pages 493–504, 1996.

[8] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. 18th Intl' Conf. on Data Engineering (ICDE)*, 2002.

[9] L. Denoyer and P. Gallinari. Bayesian network model for semi-structured document classification. *Information Processing and Management*, 40(5):807–827, 2004.

[10] L. Denoyer, P. Gallinari, and A. Vercouster. Xml mining challenge at inex 2005. Technical report, University of Paris VI, 2006.

[11] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., second edition, 2001.

[12] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltàn Szlàvic, editors. *INEX 2004 Workshop Proceedings. Dagstuhl, Germany, December 15–17, 2003*. ERCIM, 2005.

[13] M. Hagenbuchner, A. Sperduti, A.C. Tsoi, F. Trentini, F. Scarselli, and M. Gori. Clustering XML documents using self-organizing maps for structures. In Fuhr et al. [12], pages 581–496.

[14] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[15] M. Mani, D. Lee, and M. Murata. Normal forms for regular tree grammars. Technical report, UCLA Computer Science Department, 2001.

[16] A. Nierman and H. Jagadish. Evaluating structural similarity in XML documents. In *Proc. 5th Intl' Workshop on the Web and Databases (WebDB)*, June 2002.

[17] L. Peters. Change detection in XML trees: A survey. In *4th Twente Student Conference on IT*, 2005.

[18] Fabrizio Sebastiani. Machine learning in automated text categorisation. *ACM Computing Surveys*, 34(1):1–47, 2002.

[19] S. Selkow. The tree-to-tree editing problem. In *Information Processing Letters*, volume 6, pages 184–186, 1977.

[20] D. Shasha and K. Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[21] K. Tai. The tree-to-tree correction problem. *ACM Journal*, 26(3):422–433, 1979.

[22] J.T.L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 6(4):559–571, 1994.

[23] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proc. 22nd ACM SIGIR Intl' Conf. on Research and Development in IR*, pages 42–49. ACM Press, 1999.

[24] M. Zaki and C. Aggarwal. XRules: An effective structural classifier for XML data. In *Proc. 9th Intl' Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[25] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *J. on Computing*, 18(6):1245–1262, 1989.

[26] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for XML documents. In *Workshop on Knowledge and Experience Management (FGWM)*, 2003.