Efficient XML Structural Similarity Detection using Sub-tree Commonalities

Joe Tekli[1], Richard Chbeir[1], Kokou Yetongnon[1]

[1] LE2I Laboratory UMR-CNRS, University of Bourgogne
21078 Dijon Cedex France
{joe.tekli, richard.chbeir, kokou.yetongnon}@u-bourgogne.fr

Abstract. *Developing efficient techniques for comparing XML-based documents becomes essential in the database and information retrieval communities. Various algorithms for comparing hierarchically structured data, e.g. XML documents, have been proposed in the literature. Most of them make use of techniques for finding the edit distance between tree structures, XML documents being modeled as ordered labeled trees. Nevertheless, a thorough investigation of current approaches led us to identify several unaddressed structural similarities, i.e. sub-tree related similarities, while comparing XML documents. In this paper, we provide an improved comparison method to deal with such resemblances. Our approach is based on the concept of tree edit distance, introducing the notion of commonality between sub-trees. Experiments demonstrate that our approach yields better similarity results with respect to alternative methods, while maintaining quatratic time complexity.*

1. Introduction

W3C's XML (eXtensible Mark-up Language) has recently gained unparalleled importance as a fundamental standard for efficient data management and exchange. Information destined to be broadcasted over the web is henceforth represented using XML, in order to guaranty its interoperability. Owing to the unprecedented web exploitation of XML, XML-based comparison, especially for heterogeneous[1] documents, becomes a central issue in the information retrieval and database communities, its applications ranging over version control, change management and data warehousing [Chawathe *et al.* 1996] [Chawathe 1999] [Cobéna *et al.* 2002], XML query systems [Schlieder 2001] [Zhang *et al.* 2003], as well as the classification/clustering of XML documents gathered from the web against a set of DTDs declared in an XML database [Nierman and Jagadish 2002] [Bertino *et al.* 2004] [Dalamagas *et al.* 2006].

A range of algorithms for comparing semi-structured data, e.g. XML-based documents, have been proposed in the literature. Most of these approaches make use of techniques for finding the edit distance between tree structures, XML documents being treated as Ordered Labeled Trees (OLT) [WWW Consortium]. Nonetheless, a thorough investigation of the most recent and efficient XML structural similarity approaches [Chawathe 1999] [Nierman and Jagadish 2002] [Dalamagas *et al.* 2006] led us to pinpoint certain cases where the corresponding edit distance outcome is inaccurate, as we will see in the motivating examples.

[1] We note by *heterogeneous XML document*, one that does not conform to a given grammar (DTD/XML Schema), which is the case of a lot of XML documents found on the web [Nierman and Jagadish 2002].

1.1. Motivation

Consider, for example, dummy XML trees *A*, *B* and *C* in Figure 1. One can realize that tree *A* is structurally more similar to *B*, than to *C*, the sub-tree A_1, made up of nodes *b*, *c* and *d*, appearing twice in *B* (B_1 and B_2) and only once in *C* (C_1). Nonetheless, such (sub-tree) structural similarities are left unaddressed by most existing approaches, e.g. Chawathe's method [Chawathe 1999] considered as a reference point for the latest tree edit distance algorithms [Nierman and Jagadish 2002] [Dalamagas *et al.* 2006]. Chawathe's edit distance process [Chawathe 1999] permits applying changes to only one node at a time (using node *insert*, *delete* and *update* operations, with unit costs), thus yielding the same structural similarity value while comparing trees *A*/*B* and *A*/*C*.

- *Dist(A, B) = Dist(A, C) = 3*, which is the cost of three consecutive insert operations introducing nodes *b*, *c* and *d* (*e*, *f* and *g*) in tree *A* transforming it into *B* (*C*).
- Therefore, *Sim(A, B) = Sim(A, C) = 0.25* where *Sim = 1 / (1+Dist)*.

In theory, structural resemblances such as those between trees *A*/*B* and *A*/*C* could be taken into consideration by applying generalizations of Chawathe's approach [Chawathe 1999], developed in [Nierman and Jagadish 2002] and [Dalamagas *et al.* 2006] (introducing edit operations allowing the insertion and deletion of whole sub-trees). Yet, our examination of the approaches provided in [Nierman and Jagadish 2002] [Dalamagas *et al.* 2006] led us to identify certain cases where sub-tree structural similarities are disregarded:

- Similarity between trees *A*/*D* (sub-trees A_1 and D_2) in comparison with *A*/*E*.
- Similarity between trees *F*/*G* (sub-trees F_1 and G_2) relatively to *F*/*H*.
- Similarity between trees *F*/*I* (sub-tree F_1 and tree *I*) in comparison with *F*/*J*.

In fact, the authors of [Nierman and Jagadish 2002] make use of the *contained in* relation between trees (cf. Definition 2) so as to capture sub-tree similarities. Following [Nierman and Jagadish 2002], a tree *A* may be inserted in *T* only if *A* is already *contained in* the source tree *T*. Similarly, a tree *A* may be deleted only if *A* is already *contained in* the destination tree *T*. Therefore, the approach in [Nierman and Jagadish 2002] captures the sub-tree structural similarities between XML trees *A/B* in Figure 1, transforming *A* to *B* in a single edit operation: (inserting sub-tree B_2 in *A*, B_2 occurring in *A* as A_1), whereas transforming *A* to *C* would always need three consecutive insert operations (inserting nodes *e*, *f* and *g*).

Nonetheless, when the containment relation is not fulfilled, certain structural similarities are ignored. Consider, for instance, trees *A* and *D* in Figure 1. Since D_2 is not contained in *A*, it is inserted via four edit operations instead of one (insert tree), while transforming *A* to *D*, ignoring the fact that part of D_2 (sub-tree of nodes *b*, *c*, *d*) is identical to A_1. Therefore, equal distances are obtained when comparing trees *A/D* and *A/E*, disregarding *A*/*D*'s structural resemblances.

- Dist(A, D) = Cost_{Ins}(h) + Cost_{Ins}(b) + Cost_{Ins}(c) + Cost_{Ins}(d) + Cost_{Ins}(h) = 1 + 4 = 5
- Dist(A, E) = Cost_{Ins}(h) + Cost_{Ins}(e) + Cost_{Ins}(f) + Cost_{Ins}(g) + Cost_{Ins}(h) = 1 + 4 = 5

Likewise for the *D* to *A* transformation (tree D_2 will not be deleted via a single delete tree operation since it is not contained in the destination tree *A*), achieving *Dist(D, A) = Dist(E, A) = 5*. Other types of sub-tree structural similarities that are missed by [Nierman and Jagadish 2002]'s approach (and likewise missed by [Chawathe 1999] [Dalamagas *et al.* 2006]) can be identified when comparing trees *F*/*G* and *F*/*H*, as well as *F*/*I* and *F*/*J*. The *F*, *G*, *H* case is different than its predecessor (the *A*, *D*, *F* case) in that the sub-trees sharing structural similarities (F_1 and G_2) occur at different depths (whereas with *A*/*D*, A_1 and D_2 are at the same depth). On the other hand, the *F*, *I*, *G* case differs from the previous ones since single level trees (*I* and *J*), which only encompass leaf nodes, are implicated in the comparison process.

Please note that [Dalamagas *et al.* 2006]'s algorithm yields the same results as [Nierman and Jagadish 2002]'s algorithm, in the above examples, which is why it is not discussed in detailed (it is as a specialized version of [Nierman and Jagadish 2002] where tree insertion/deletion costs are computed as the sum of the costs of inserting/deleting all individual nodes in the considered sub-trees).

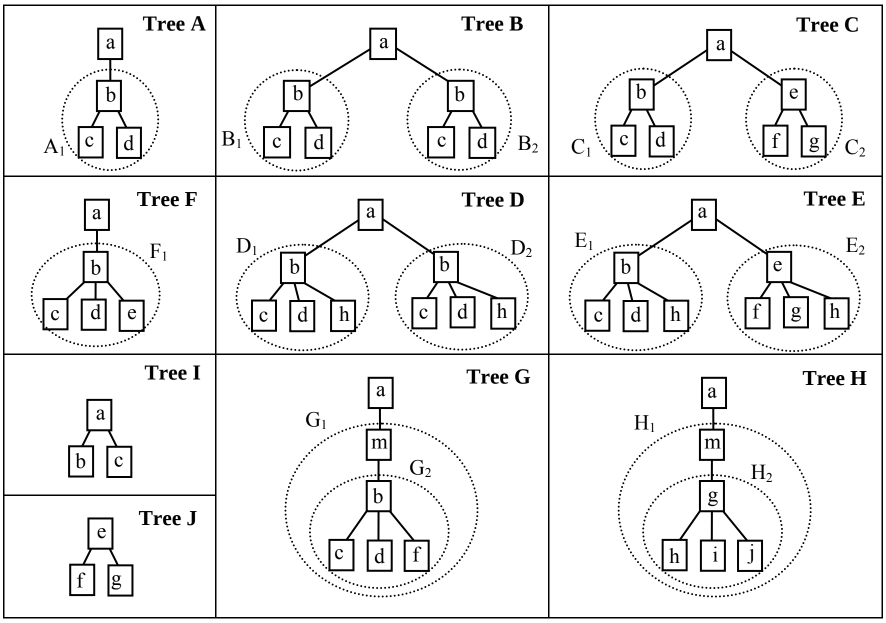


Figure 1. Sample XML trees.

1.2. Contribution and Organization of the Paper

The goal of our study here is to provide an improved XML structural similarity method for comparing heterogeneous XML documents. In short, we aim to build on existing approaches, mainly [Chawathe 1999] [Nierman and Jagadish 2002], in order to take into account the various sub-tree structural commonalities while comparing XML trees. The contribution of the paper can be summarized as follows: i) introducing the notion of *structural commonality* between sub-trees, putting forward an algorithm for its discovery, ii) introducing an efficient algorithm for computing tree-based edit operations costs able to consider, via the sub-tree *commonality* notion, XML sub-tree structural similarities, iii) developing a prototype to evaluate and validate our approach. The remainder of this paper is organized as follows. Section 2 reviews background in XML structural similarity. Section 3 presents preliminary definitions. In Section 4, we develop our XML structural similarity approach. Section 5 presents experimental evaluation results. Conclusions and ongoing work are covered in Section 6.

2. Related Work

Various methods, for determining structural similarities between hierarchically structured data, particularly XML documents, have been proposed. Most of them derive, in one way or another, the dynamic programming techniques for finding edit distance between strings [Levenshtein 1966] [Wagner and Fisher 1974] [Wong and Chandra 1976]. In essence, all these approaches aim at finding the cheapest sequence of edit operations that can transform one tree into another. Nevertheless, tree edit distance algorithms can be distinguished by the set of edit operations that are allowed as well as overall complexity and performance levels.

Early approaches in [Zhang and Shasha 1989] [Shasha and Zhang 1995] allow insertion, deletion and relabeling of nodes anywhere in the tree. Yet, they are relatively complex. For instance, the approach in [Shasha and Zhang 1995] has a time complexity $O(|A||B|\ depth(A)$

depth(B)) (|*A*| and |*B*| denote tree cardinalities while *depth(A)* and *depth(B)* are the depths of the trees). [Chawathe *et al.* 1996] [Cobéna *et al.* 2002] restrict insertion and deletion operations to leaf nodes and add a move operator that can relocate a sub-tree, as a single edit operation, from one parent to another. However, corresponding algorithms do not guaranty optimal results. Recent work in [Chawathe 1999] restricts insertion and deletion operations to leaf nodes, and allows the relabeling of nodes anywhere in the tree, while disregarding the move operation. The overall complexity of [Chawathe 1999]'s algorithm is of $O(N^2)$. [Nierman and Jagadish 2002] extend the approach in [Chawathe 1999] by adding two new operations: insert tree and delete tree to allow insertion and deletion of whole sub-trees within in an Ordered Labeled Tree. [Nierman and Jagadish 2002]'s overall complexity simplifies to $O(N^2)$ despite being conceptually more complex than its predecessor. A specialized version of [Nierman and Jagadish 2002]'s algorithm is provided in [Dalamagas *et al.* 2006]. On the other hand, an original structural similarity approach is presented in [Flesca 2002]. It disregards OLTs and utilizes the Fast Fourier Transform to compute similarity between XML documents. Yet, the authors did not compare their algorithm's optimality to existing edit distance approaches. Another approach, disregarding edit distance computations was introduced by [Sanz *et al.* 2005]. It utilizes specific indexing structures rather than tree edit distance. Experimental results in [Sanz *et al.* 2005] show that the approach is of linear complexity. Nonetheless, the authors of [Sanz *et al.* 2005] did not compare their algorithm's optimality to existing approaches.

3. Basic Definitions

Def. 1 - Ordered Labeled Tree: it is a rooted tree in which the nodes are ordered and labeled. We note by *λ(T)* the label of the root node of tree *T*. In the rest of this paper, the term *tree* means *rooted ordered labeled tree*.

Def. 2 - Tree "*Contained in*" relationship: a tree *A* is said to be *contained in* a tree *T* if all nodes of *A* occur in *T*, with the same parent/child edge relationship and node order. Additional nodes may occur in *T* between nodes in the embedding of *A* (e.g., tree *J* is *contained in* tree *E*).

Def. 3 - Sub-tree: given two trees *T* and *T'*, *T'* is a sub-tree of *T* if all nodes of *T'* occur in *T*, with the same parent/child edge relationship and node order, such as no additional nodes occur in the embedding of *T'* (e.g., tree *J* in Figure 1 is a sub-tree of *C*, whereas *J* does not qualify as a sub-tree of *E* since node *h* occur in its embedding in *E*).

Def. 4 - Ld-pair representation of a node: it is defined as the pair *(l, d)* where: *l* and *d* are respectively the node's label and depth in the tree. We use *p.l* and *p.d* to refer to the label and the depth of an *ld-pair* node *p* respectively.

A_1 = ((b, 0), (c, 1), (d, 1))	D_1 = ((b, 0), (c, 1), (d, 1), (h, 1))	G_1 = ((m, 0), (b, 1), (c, 2), (d, 2), (e, 2))
B_1 = ((b, 0), (c, 1), (d, 1))	D_2 = ((b, 0), (c, 1), (d, 1), (h, 1))	G_2 = ((b, 0), (c, 1), (d, 1), (f, 1))
B_2 = ((b, 0), (c, 1), (d, 1))	E_1 = ((b, 0), (c, 1), (d, 1), (h, 1))	H_1 = ((m, 0), (g, 1), (h, 2), (i, 2), (j, 2))
C_1 = ((b, 0), (c, 1), (d, 1))	E_2 = ((e, 0), (f, 1), (g, 1), (h, 1))	H_2 = ((g, 0), (h, 1), (i, 1), (j, 1))
C_2 = ((e, 0), (f, 1), (g, 1))	F_1 = ((b, 0), (c, 1), (d, 1), (e, 1))	

Figure 2. Ld-pair representations of all sub-trees in XML trees *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*[1] in Figure 1.

Def. 5 - Ld-pair representation of a tree: the *ld-pair* representation of a tree is the list, in preorder, of the *ld-pairs* of its nodes (cf. Figure 2). Given a tree in *ld-pair* representation $T = (t_1, t_2, ..., t_n)$, *T[i]* refers to the i^{th} node t_i of *T*. Consequently, *T[i].l* and *T[i].d* denote, respectively, the label and the depth of the i^{th} node of *T*, *i* designating the preorder traversal rank of node *T[i]* in *T*.

Def. 6 - Structural commonality between sub-trees: given two sub-trees $A = (a_1, ..., a_m)$ and $B = (b_1, ..., b_n)$, the structural commonality between *A* and *B*, designated by *ComSubTree(A, B)*,

[1] Trees *I* and *J* only encompass leaf nodes which is why they are not considered in this example.

is a set of nodes $N = \{n_1, ..., n_p\}$ such that $\forall\, n_i \in N$, n_i occurs in A and B with the same label, depth and relative node order (in preorder traversal ranking) as in A and B. For $1 \leq i \leq p$; $1 \leq r \leq m$; $1 \leq u \leq n$:

(1)$n_i.l = a_r.l = b_u.l$

(2)$n_i.d = a_r.d = b_u.d$

(3)For any $n_j \in N \,/\, i \leq j$, $\exists\; a_s \in A$ and $b_v \in B$ such as:

- $n_j.l = a_s.l = b_v.l$
- $n_j.d = a_s.d = b_v.d$
- $r \leq s,\ u \leq v$

(4)There is no set of nodes N' that satisfies conditions *1*, *2* and *3* and is of larger cardinality than N.

In other words, *ComSubTree(A, B)*[1] identifies the set of matching nodes between sub-trees *A* and *B*, node matching being undertaken with respect to node label, depth and relative preorder ranking. Please note that in the rest of the paper, the term *commonality* always stands for the structural commonality.

On the other hand, our edit distance XML structural similarity approach utilizes five edit operations, adopted from [Chawathe 1999] [Nierman and Jagadish 2002]: *node insertion*, *node deletion* and n*ode update*, as well as *tree insertion* and *tree deletion*. Nonetheless, due to lack of space, corresponding formal definitions are disregarded.

4. Proposal

Our XML structural similarity approach consists of two algorithms: i) an algorithm for identifying the *Commonality Between two Sub-trees* (*CBS*), ii) and an algorithm for computing the *Tree edit distance Operations Costs* (*TOC*), making use of *CBS*, its results being exploited via [Nierman and Jagadish 2002]'s main edit distance algorithm in order to identify the structural similarity between two XML documents (cf. Figure 3).

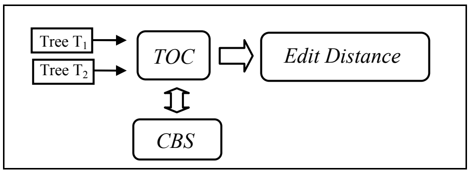


Figure 3. Simplified activity diagram of our XML structural similarity approach.

4.1. Commonality Between Sub-trees *(CBS)*

In order to capture the sub-tree structural similarities unaddressed by [Nierman and Jagadish 2002]'s approach, we identify the need to replace the tree *contained in* relation making up a necessary condition for executing tree insertion and deletion operations in [Nierman and Jagadish 2002], by introducing the notion of *commonality* between two sub-trees. Following Definition 6, the problem of finding the structural commonality between two sub-trees SbT_i and SbT_j is equivalent to finding the maximum number of matching nodes in SbT_i and SbT_j ($|ComSubTree(SbT_i, SbT_j)|$). On the other hand, the problem of finding the shortest edit distance between SbT_i and SbT_j comes down to identifying the minimal number of edit operations that can transform SbT_i to SbT_j. Those are dual problems since identifying the shortest edit distance between two sub-trees (trees) underscores, in a roundabout way, their maximum number of matching nodes.

[1] Our sub-tree structural commonality definition can be equally applied to whole trees (a sub-tree being basically a tree). However, in this study, it is mostly utilized with sub-trees.

Therefore, we introduce in Figure 4 an algorithm (*CBS*), based on the edit distance concept, to identify the structural commonality between sub-trees (similarly to [Myers 1986] in which Myers develops an edit distance based approach for computing the longest common sub-sequence between two strings). Note that in *CBS*, sub-trees are treated in their *ld-pair* representations (cf. Figure 2). Using the *ld-pair* tree representations, sub-trees are transformed into *modified* sequences (*ld-pairs*), making them suitable for *standard* edit distance computations.

Afterward, the maximum number of matching nodes between SbT_i and SbT_j, $|ComSubTree(SbT_i, SbT_j)|$, is identified with respect to the computed minimum edit distance:

- Total number of deletions - we delete all nodes of SbT_i except those having matching nodes in SbT_j: $\sum_{Deletions} = |SbT_i| - |ComSubTree(SbT_i, SbT_j)|$
- Total number of insertions - we insert into SbT_i all nodes of SbT_j except those having matching nodes in SbT_i: $\sum_{Insertions} = |SbT_j| - |ComSubTree(SbT_i, SbT_j)|$
- Following *CBS*, using constant unit costs (=1) for node insertion and deletion operations, the edit distance between sub-trees SbT_i and SbT_j becomes as follows: $Dist[|SbT_i|][|SbT_j|]$ $= \sum_{Deletions} \times 1 + \sum_{Insertions} \times 1 = |SbT_i| + |SbT_j| - 2\,|ComSubTree(SbT_i, SbT_j)|$
- Therefore, $|ComSubTree(SbT_i, SbT_j)| = \dfrac{|SbT_i| + |SbT_j| - Dist[|SbT_i|][|SbT_j|]}{2}$

```
Algorithm CBS()
Input: Sub-trees $SbT_i$ and $SbT_j$ (in ld-pair representations)
Output: $|ComSubTree(SbT_i, SbT_j)|$
Begin
    Dist [][] = new [0...$|SbT_i|$][0...$|SbT_j|$]
    Dist[0][0] = 0
    For (n = 1 ; n ≤ $|SbT_i|$ ; n++)
        { Dist[n][0] = Dist[n-1][0] + $Cost_{Del}$($SbT_i$[n]) }
    For (m = 1 ; m ≤ $|SbT_j|$ ; m++)
        { Dist[0][m] = Dist[0][m-1] + $Cost_{Ins}$($SbT_j$[m]) }
    For (n = 1 ; n ≤ $|SbT_i|$ ; n++)
    {
      For (m = 1 ; m ≤ $|SbT_j|$ ; m++)
      {
        Dist[n][m] = min{
              If ($SbT_i$[n].d = $SbT_j$[m].d & $SbT_i$[n].l = $SbT_j$[m].l)
                  { Dist[n-1][m-1] },
              Dist[n-1][m] + $Cost_{Del}$($SbT_i$[n]),    // simplified node
              Dist[n][m-1] + $Cost_{Ins}$($SbT_j$[m])     // operations syntaxes.
                           }
      }
    }
    Return $\frac{|SbT_i|+|SbT_j| - Dist[|SbT_i|][|SbT_j|]}{2}$    // $|ComSubTree(SbT_i, SbT_j)|$
End
```

Figure 4. Algorithm *CBS* for identifying the structural commonality between sub-trees.

For instance, $|ComSubTree(A_1,D_1)|$=3 (nodes *b*, *c*, *d*), $|ComSubTree(E_2,G_2)|$=1 (node *f*).

4.2. Tree Edit Operations Costs (*TOC*)

As stated previously, *TOC* is an algorithm dedicated to computing the tree edit distance operations costs. These costs will be exploited via [Nierman and Jagadish 2002]'s main edit distance approach (cf. Figure 6) providing an improved and more accurate XML structural similarity measure. *TOC*, developed in Figure 5, consists of three main steps:

- Step 1 (lines 2-13) identifies the structural commonalities between each pair of non leaf sub-trees in the source and destination trees respectively (T_1 and T_2), assigning tree insert/delete operation costs accordingly.
- Step 2 (lines 14-18) identifies the structural commonalities between each non leaf node sub-tree in the source tree (T_1) and the destination tree (T_2) as a whole, updating delete tree operation costs correspondingly.
- Step 3 (lines 19-24) identifies the structural commonalities between each non leaf node sub-tree in the destination tree (T_2) and the source tree (T_1) as a whole, modifying insert tree operation costs accordingly.

Note that the relevance of steps 2 and 3 becomes obvious when single level trees (trees made of leaf nodes) are involved in the comparison process (the *F*, *I*, *J* case discussed in Section 3.3). The insert/delete tree operations costs corresponding to leaf node sub-trees are not computed in *TOC* since such sub-trees come down to single nodes. Inserting/deleting a leaf node sub-tree is ultimately undertaken via simple node insertion/deletion operations which are assigned constant unit costs (=1). Using *CBS*, *TOC* identifies the structural *commonality* between each and every pair of sub-trees (SbT_i, SbT_j) in the two trees *A* and *B* being compared (step 1), as well as their commonalities with the whole trees A and B, respectively (steps 2 and 3).

Consequently, those values are normalized via corresponding tree/sub-tree cardinalities $Max(|SbT_i|\ ,\ |SbT_j|)$ to be comprised between 0 and 1:

- $\frac{CBS(\text{SbT}_i, \text{SbT}_j)}{\text{Max}(|\text{SbT}_i|, |\text{SbT}_j|)} = 0$ When there is no structural commonality between SbT_i and SbT_j : $CBS(\text{SbT}_i, \text{SbT}_j) = 0$.
- $\frac{CBS(\text{SbT}_i, \text{SbT}_j)}{\text{Max}(|\text{SbT}_i|, |\text{SbT}_j|)} = 1$ When the sub-trees are identical: $CBS(\text{SbT}_i, \text{SbT}_j) = |\text{SbT}_i| = |\text{SbT}_j|$

For instance, $\frac{CBS(\text{A}_1, \text{D}_1)}{\text{Max}(|\text{A}_1|, |\text{D}_1|)} = \frac{3}{4} = 0.75$, $\frac{CBS(\text{E}_2, \text{G}_2)}{\text{Max}(|\text{E}_2|, |\text{G}_2|)} = \frac{1}{4} = 0.25$ (cf. Figure 1).

Thus, using the normalized commonality, tree operations costs would vary as follows:

Maximum insert/delete tree cost for sub-tree Sb_i:

$$\text{Cost}_{\text{InsTree/DelTree}}(\text{Sb}_i) = \sum_{\text{All nodes } x \text{ of SbT}_i} \text{Cost}_{\text{Ins/Del}}(x) \times 1$$

Minimum insert/delete tree cost for sub-tree Sb_i:

$$\text{Cost}_{\text{InsTree/DelTree}}(\text{Sb}_i) = \sum_{\text{All nodes } x \text{ of SbT}_i} \text{Cost}_{\text{Ins/Del}}(x) \times \frac{1}{2}$$

Following *TOC*, the maximal insert/delete tree operation cost for a given sub-tree SbT_i (attained when no sub-tree structural similarities with SbT_i are identified in the source/destination tree respectively) is the sum of the costs of inserting/deleting every individual node of Sb_i. The minimal insert/delete tree operation cost for SbT_i (attained when a sub-tree structurally identical to SbT_i is identified in the source/destination tree respectively) is equal to half its corresponding insert/delete tree maximum cost. The minimal tree operation cost is defined in such a way in order to guaranty that the cost of inserting/deleting a tree will never be less than the cost of inserting/deleting a single node (single node operations having unit costs). In fact, *TOC* is based on the intuition that tree operations are more costly than node operations.

Proof: The smallest sub-tree that can be treated via a tree operation is a sub-tree consisting of two nodes. For such a tree, the minimum insert/delete tree operation cost would be equal to 1 (its maximum cost being equal to 2), equivalent to the cost of inserting/deleting a single node, which is the lowest tree operation cost attainable following TOC.

```
Algorithm TOC()
Input: Trees $T_1$ and $T_2$
Output: Insert tree and delete tree operations costs
Begin
    For each sub-tree $SbT_i$ in $T_1$ / $|SbT_i| > 1$          // Excluding leaf
    {                                                        // node sub-trees in $T_1$.
      $Cost_{DelTree}(SbT_i) = \sum_{\text{All nodes } x \text{ of } SbT_i} Cost_{Del}(x)$
      For each sub-tree $SbT_j$ in $T_2$ / $|SbT_j| > 1$        // Excluding leaf
      {                                                      // node sub-trees in T2.
          $Cost_{InsTree}(SbT_j) = \sum_{\text{All nodes } x \text{ of } SbT_j} Cost_{Ins}(x)$
          $Cost_{DelTree}(SbT_i) = Min\{ Cost_{DelTree}(SbT_i),$
                $\sum_{\text{All nodes } x \text{ of } SbT_i} Cost_{Del}(x) \times \frac{1}{1 + \frac{CBS(SbT_i, SbT_j)}{Max(|SbT_i|, |SbT_j|)}} \}$
          $Cost_{InsTree}(SbT_j) = Min\{ Cost_{InsTree}(SbT_j),$
                $\sum_{\text{All nodes } x \text{ of } SbT_j} Cost_{Ins}(x) \times \frac{1}{1 + \frac{CBS(SbT_i, SbT_j)}{Max(|SbT_i|, |SbT_j|)}} \}$
      }
    }
    For each sub-tree $SbT_i$ in $T_1$ / $|SbT_i| > 1$          // Excluding leaf
    {                                                        // node sub-trees in $T_1$.
      $Cost_{DelTree}(SbT_i) = Min\{ Cost_{DelTree}(SbT_i),$
            $\sum_{\text{All nodes } x \text{ of } SbT_i} Cost_{Del}(x) \times \frac{1}{1 + \frac{CBS(SbT_i, T_2)}{Max(|SbT_i|, |T_2|)}} \}$
    }
    For each sub-tree $SbT_j$ in $T_2$ / $|SbT_j| > 1$          // Excluding leaf
    {                                                        // node sub-trees in $T_2$.
      $Cost_{InsTree}(SbT_j) = Min\{ Cost_{InsTree}(SbT_j),$
            $\sum_{\text{All nodes } x \text{ of } SbTj} Cost_{Ins}(x) \times \frac{1}{1 + \frac{CBS(T_1, SbT_j)}{Max(|T_1|, |SbT_j|)}} \}$
    }
End
```

Figure 5. Tree edit distance Operations Costs algorithm.

Using *TOC*, we compute the costs of tree insertion and deletion operations based on their corresponding trees' maximum normalized commonality values (a maximum commonality value inducing a minimum tree operation cost). Therefore, instead of utilizing the *contained in* relation introduced in [Nierman and Jagadish 2002] (cf. Definition 2) in order to permit or deny tree insertion/deletion operations (thus disregarding certain sub-tree structural similarities while comparing two XML trees as shown in Section 3.3), we permit the insertion and deletion of any/all sub-trees by varying their corresponding tree insertion/deletion operation costs with respect to their structural similarities with the source/destination trees/sub-trees respectively. Note that inserting/deleting the whole destination/source trees is not allowed in our approach. In fact, by rejecting such operations, one cannot delete the entire source tree in one step and insert the entire destination tree in a second step, which completely undermine the purpose of the insert/delete tree operations.

```
Algorithm EditDistance()

Input: Trees A and B
Output: Edit distance between A and B

Begin
    M = Degree(A)                // The number of first level sub-trees in A.
    N = Degree(B)                // The number of first level sub-trees in B.
    Dist [][] = new [0...M][0…N]
    Dist[0][0] = $Cost_{Upd}(\lambda(A), \lambda(B))$
    For (i = 1 ; i ≤ M ; i++) { Dist[i][0] = Dist[i-1][0] + $Cost_{DelTree}(A_i)$ }
    For (j = 1 ; j ≤ N ; j++) { Dist[0][j] = Dist[0][j-1] + $Cost_{InsTree}(B_j)$ }
    For (i = 1 ; i ≤ M ; i++)
    {
      For (j = 1 ; j ≤ N ; j++)
      {
        Dist[i][j] = min{
                  Dist[i-1][j-1] + EditDistance($A_i$, $B_j$),     //Dynamic
                  Dist[i-1][j] + $Cost_{DelTree}(A_i)$,             //programming
                  Dist[i][j-1] + $Cost_{InsTree}(B_j)$
                      }
      }
    }
    Return Dist[M][N]
End
```

Figure 6. Edit distance algorithm [Nierman and Jagadish 2002].

4.3. Computation Examples

Due to space limitations, we only detail the edit distance computations when comparing XML documents *A*, *D* and *E*. For the remaining cases, results are reported in Table 1. Recall that trees *D* and *E* are considered similar with respect to *A* following current approaches, i.e. [Chawathe 1999] [Nierman and Jagadish 2002] [Dalamagas *et al.* 2006], despite the fact that *A/D* share more structural similarities than *A/E* (as discussed in Section 3.3). In order to compare trees *A*/*D*, we start by executing algorithm *TOC* which yields the following insertion/deletion operations costs. When applied to XML trees *A* and *D*, our approach yields *EditDistance(A, D) = 3.2856* (cf. Table 1) having:

$$\text{Cost}_{\text{DelTree}}(\text{A}_1) = \sum_{\text{All nodes } x \text{ of } \text{A}_1} \text{Cost}_{\text{Del}}(x) \times \frac{1}{1 + \dfrac{CBS(\text{A}_1, \text{D}_1)}{\text{Max}(|\text{A}_1|, |\text{D}_1|)}} = 3 \times \frac{1}{1+0.75} = 1.7143$$

Likewise, $\text{Cost}_{\text{InsTree}}(\text{D}_1) = \text{Cost}_{\text{InsTree}}(\text{D}_2) = 4 \times 1/(1+0.75) = 2.2856$

Table 1. Computing edit distance for XML trees *A* and *D*.

	λ(D)	D_1	D_2
λ(A)	0	2.2856	4.5712
A_1	1.7143	1	3.2856

- Dist[1, 1] = 1: cost of transforming A_1 to D_1 (inserting node h).
- Dist[1, 2] = 2.2856 + Dist[1, 1] = 3.2856: inserting D_2 into A.

On the other hand, when applied to XML trees *A* and *E*, our approach yields *EditDistance(A, E) = 5* (cf. Table 2) having:

$$\text{Cost}_{\text{DelTree}}(\text{A}_1) = \sum_{\text{All nodes } x \text{ of A}_1} \text{Cost}_{\text{Del}}(x) \times \frac{1}{1 + \dfrac{CBS(\text{A}_1, \text{E}_1)}{\text{Max}(|\text{A}_1|, |\text{E}_1|)}} = 3 \times \frac{1}{1+0.75} = 1.7143$$

Likewise, $\text{Cost}_{\text{InsTree}}(\text{E}_1)$=4 × 1/(1+0.75)=2.2856 and $\text{Cost}_{\text{InsTree}}(\text{E}_2)$=4 × 1/(1+0)=4

Table 2. Computing edit distance for XML trees *A* and *E*.

	0	E_1	E_2
0	0	2.2856	6.2856
A_1	1.7143	1	5

- Dist[1, 1] = 1: transforming A_1 into E_1 (inserting node h).
- Dist[1, 2] = 4 + Dist[1, 1] = 5: cost of inserting E_2 into A.

Therefore, our approach is able to efficiently compare XML documents *A*, *D* and *E* underlining that documents *A/D* are more similar than *A/E* (pointing out structural similarities that are not detected via existing approaches):

- Sim(A/D) = 1/(1+Dist(A, D)) = 1 /(1 + 3.2836) = 0.2333
- Sim(A/E) = 1/(1+Dist(A, E)) = 1 /(1 + 5) = 0.1667

As for XML trees *A*, *D* and *E*, our approach detects the structural similarities between *A/B* (with respect to *A/C*), *F/G* (with respect to *F/H*), as well as between *F/I* (with respect to *F/J*). Results are reported in Table 3.

Table. 3. Distance/similarity values attained using our comparison approach for the various XML comparison examples treated throughout the paper.

	Our Approach		Nierman & Jagadish.	Dalamagas *et al.*	Chawathe
	Distance	Similarity			
A/B	1.5	0.4	*Detected*	*Not detected*	*Not detected*
A/C	3	0.25			
A/D	3.2856	0.2333	*Not detected*	*Not detected*	*Not detected*
A/E	5	0.1667			
F/G	5.4106	0.1560	*Not detected*	*Not detected*	*Not detected*
F/H	7	0.125			
F/I	5.2856	0.1591	*Not detected*	*Not detected*	*Not detected*
F/J	6	0.1429			

4.4. Overall Complexity

The overall complexity of our approach simplifies to $O(|T_1||T_2|)$:

- Our *CBS* algorithm for the identification of the commonality between two sub-trees is of complexity: $O(|SbT_i||SbT_j|)$ where $|SbT_i|$ and $|SbT_j|$ denote the cardinalities of the compared sub-trees.
- Our *TOC* algorithm for computing the costs of tree insertion/deletion operations is of complexity $O(|T_1||T_2|)$ (encompassing *CBS* 's complexity):

$$
\begin{aligned}
&O\Big(\sum_{i=1}^{|T_1|-n_{T_1}-1} \sum_{j=1}^{|T_2|-n_{T_2}-1} |SbT_i|\,|SbT_j| + \sum_{i=1}^{|T_1|-n_{T_1}-1} |SbT_i|\,|T_2| + \sum_{j=1}^{|T_2|-n_{T_2}-1} |SbT_j|\,|T_1| \Big) \\
&= O\Big(\sum_{i=1}^{|T_1|-n_{T_1}-1} |SbT_i| \sum_{j=1}^{|T_2|-n_{T_2}-1} |SbT_j| + |T_2| \sum_{i=1}^{|T_1|-n_{T_1}-1} |SbT_i| + |T_1| \sum_{j=1}^{|T_2|-n_{T_2}-1} |SbT_j| \Big) \\
&\leq O(|T_1||T_2|)
\end{aligned}
$$

where:

- n_{T_1} and n_{T_2} represent the number of leafs in T_1 and T_2 (the compared trees)
- SbT_i and SbT_j underline sub-trees of T_1 and T_2 respectively.
- $|T_1| - n_{T_1} - 1$ designates the number of sub-trees in T_1 that do not consist of leaf nodes (similarly for $|T_1| - n_{T_1} - 1$ and the destination tree T_2).

– The edit distance algorithm (cf. Figure 6), which utilizes the results attained by *TOC* (tree operation costs), is of complexity $O(|T_1||T_2|)$.

5. Experimental Evaluation

5.1. Evaluation Metrics

In order to validate our structural similarity approach and compare its relevance with alternative methods, we make use of structural clustering. In our experiments, we adopt the well known single link hierarchical clustering techniques [Gower and Ross 1969][Halkidi *et al.* 2001] although any form of clustering could be utilized.

In order to evaluate clustering quality, we utilize *precision* and *recall* metrics introduced in [Dalamagas *et al.* 2006]. Having an a priori knowledge of which documents should be members of the appropriate cluster (mapping between original DTD clusters and the extracted clusters), the authors in [Dalamagas *et al.* 2006] define *precision* PR and *recall* R as:

$$
PR = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i} \qquad\qquad R = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} c_i}
$$

where:

– n is the total number of clusters in the clustering set considered

– a_i is the number of XML documents in C_i that indeed correspond to DTD_i (correctly clustered documents).

– b_i is the number of XML documents in C_i that do not correspond to DTD_i (mis-clustered documents).

– c_i is the number of XML documents not in C_i, although they correspond to DTD_i (documents that should have been clustered in C_i).

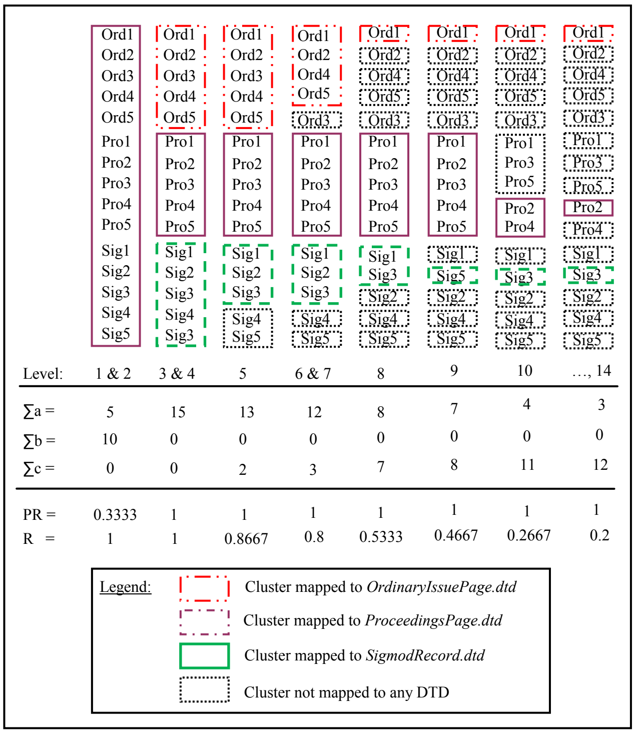
Nonetheless, in addition to comparing one approach's precision improvement to another's recall improvement, it is a common practice to compare F-values, *F-value* = $2 \times PR \times R/(PR+R)$. Therefore, as with traditional information retrieval evaluation, high *precision* and *recall*, and thus high *F-value* (indicating in our case excellent clustering quality) characterize a good similarity method.

5.2. Clustering XML Documents

In each of our experiments, we compute a series of *PR*/*R* doublets, varying the clustering level (similarity threshold) in the *[0, 1]* interval. In other words, we construct a dendrogram (cf. Figure 7) such as:

- For the initial clustering level $s_1=0$ (or s_1= the minimum similarity value attainable between any pair of documents), all XML documents appear in one global cluster, *the starting cluster*.
- For the final clustering level $s_n=1$ (with n the total number of levels, i.e. number of clustering sets in the dendrogram), each XML document will appear in a distinct cluster (to the exception of identical documents, which will remain in the same corresponding cluster).
- Intermediate clustering sets will be identified for levels s_i where $s_1<s_i<s_n$.

Then, we compute *precision* and *recall* values for each clustering set identified in the dendrogram, thus constructing their corresponding graphs that describe the system's evolution throughout the clustering process. Overall average *precision/recall* values: *Ave(PR)* and *Ave(R)* (consequently *Ave(F-Value)*) considering the whole dendrogram, are computed on the basis of the attained series, providing yet another indicator of clustering quality (structure-based comparison quality) for the comparison method being tested. A sample dendrogram with detailed *precision* and *recall* computations, underlining the clustering evolution of 15 XML documents of the ACM SIGMOD Record[1] (5 sampled from each of the *OrdinaryIssuePage.dtd*, *ProceedingsPage.dtd* and *SigmodRecord.dtd* DTD definitions), is given in Figure 7.


Ord1 Ord2 Ord3 Ord4 Ord5 Pro1 Pro2 Pro3 Pro4 Pro5 Sig1 Sig2 Sig3 Sig4 Sig5
Ord1 Ord2 Ord3 Ord4 Ord5 | Pro1 Pro2 Pro3 Pro4 Pro5 | Sig1 Sig2 Sig3 Sig4 Sig3
Ord1 Ord2 Ord3 Ord4 Ord5 | Pro1 Pro2 Pro3 Pro4 Pro5 | Sig1 Sig2 Sig3 | Sig4 Sig5
Ord1 Ord2 Ord4 Ord5 | Ord3 | Pro1 Pro2 Pro3 Pro4 Pro5 | Sig1 Sig2 Sig3 | Sig4 | Sig5
Ord1 | Ord2 | Ord4 | Ord5 | Ord3 | Pro1 Pro2 Pro3 Pro4 Pro5 | Sig1 Sig3 | Sig2 | Sig4 | Sig5
Ord1 | Ord2 | Ord4 | Ord5 | Ord3 | Pro1 Pro2 Pro3 Pro4 Pro5 | Sig1 | Sig5 | Sig2 | Sig4 | Sig5
Ord1 | Ord2 | Ord4 | Ord5 | Ord3 | Pro1 Pro3 Pro5 | Pro2 Pro4 | Sig1 | Sig3 | Sig2 | Sig4 | Sig5
Ord1 | Ord2 | Ord4 | Ord5 | Ord3 | Pro1 | Pro3 | Pro5 | Pro2 | Pro4 | Sig1 | Sig3 | Sig2 | Sig4 | Sig5
Legend:
Cluster mapped to *OrdinaryIssuePage.dtd*
Cluster mapped to *ProceedingsPage.dtd*
Cluster mapped to *SigmodRecord.dtd*
Cluster not mapped to any DTD

Level:	1 & 2	3 & 4	5	6 & 7	8	9	10	…, 14
∑a =	5	15	13	12	8	7	4	3
∑b =	10	0	0	0	0	0	0	0
∑c =	0	0	2	3	7	8	11	12
PR =	0.3333	1	1	1	1	1	1	1
R =	1	1	0.8667	0.8	0.5333	0.4667	0.2667	0.2

Figure 7. Dendrogram and detailed *PR*/*R* computations when clustering (using our structural comparison approach) 15 XML documents sampled from the SIGMOD record.

[1] Available at http://www.acm.org/sigmod/xml.

5.3. Experimental Results

We conducted experiments on real and synthetic XML documents. Two sets of 600 documents were generated from 20 real-case[1] and synthetic DTDs, using an adaptation of the IBM XML documents generator[2]. We varied the *MaxRepeats* parameter to determine the number of times a node will appear as a child of its parent node. For a real dataset, we considered the online version of the ACM SIGMOD Record. We experimented on a set of 104 documents corresponding to *OrdinaryIssuePage.dtd* (30 documents), *ProceedingsPage.dtd* (47 documents) and *SigmodRecord.dtd* (27 documents)[3].

Precision, *recall* and *F-value* graphs are presented in Figures 8, 9 and 10. Corresponding *Ave(PR)*, *Ave(R)* and *Ave(F-value)* values are reported in Table 4.

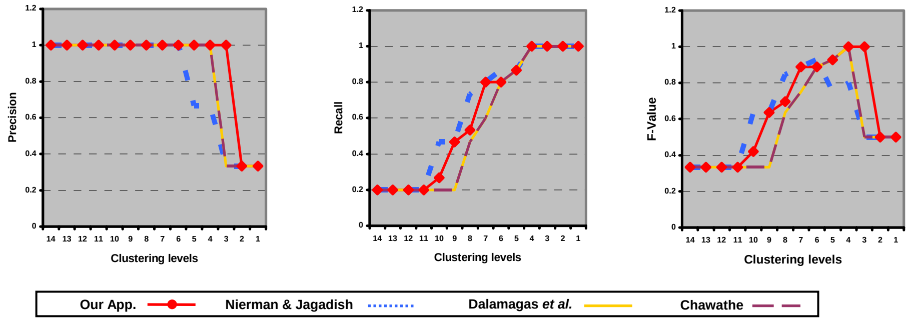


Figure 8. *PR,R, F-Value* graphs for clustering real SIGMOD Record XML documents.

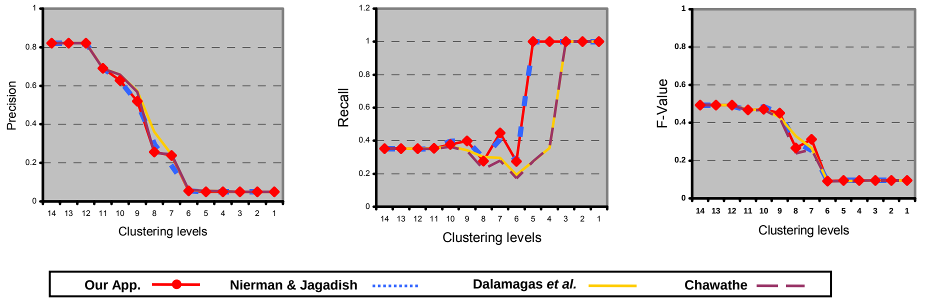


Figure 9. *PR,R, F-Value* graphs for clustering XML documents of synthetic set 1 (MaxRepeats = 5).

Table 4. Average *PR*, *R* and *F-values* obtained by varying the clustering level between [0, 1].

	SIGMOD			*Set 1 (MaxRepeats=5)*			*Set 2 (MaxRepeats =10)*		
	PR	*R*	*F-value*	*PR*	*R*	*F-value*	*PR*	*R*	*F-value*
Nierman & Jagadish	0.8095	0.6429	0.7165	0.3624	0.5840	0.4474	0.3349	0.3197	0.3271
Dalamagas *et al.*	0.8571	0.5667	0.6823	0.3788	0.4671	0.4184	0.3312	0.2844	0.3061
Chawathe	0.8571	0.5667	0.6823	0.3707	0.4581	0.4098	0.3320	0.2788	0.3031
Our approach	**0.9048**	**0.6095**	**0.7253**	**0.3644**	**0.5845**	**0.4489**	**0.3359**	**0.3624**	**0.3487**

Results, with respect to all three data sets, indicate that our approach yields improved global clustering quality (i.e. structural comparison quality) in comparison with current alternative approaches. To further validate our approach, a set of experimental tests on more complex structure document corpus including macromolecular tree patterns encoded in XML (e.g. RNA structures) is currently ongoing. It is worth noting that our experiments will not

[1] From http://www.xmlfiles.com and http://www.w3schools.com

[2] http://www.alphaworks.ibm.com.

[3] We were able to find only one XML file conforming to *SigmodRecord.dtd*: *SigmodRecord.xml*. However, due to its relatively large size (479KB) in comparison with the XML documents corresponding to the other two DTDs (10KB of average size per document), we carefully decomposed *SigmodRecord.xml* to 27 documents, creating a set of XML documents conforming to *SigmodRecord.dtd*.

address the current INEX (INitiative for the Evaluation of XML Retrieval) XML corpus since its collections present little heterogeneity in both the tags and structures (INEX focuses mainly on text-rich documents).

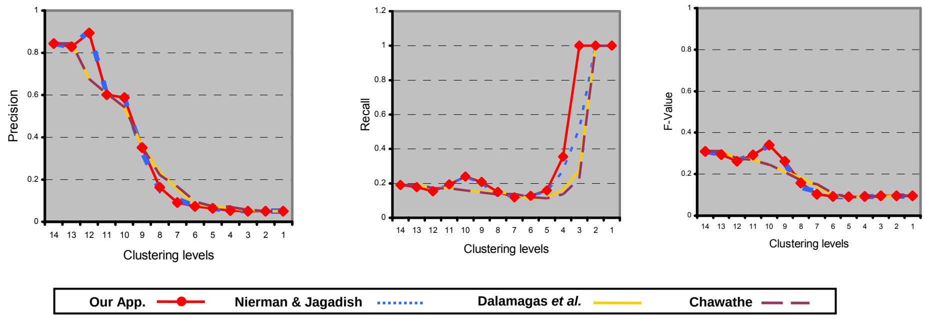


Figure 10. *PR,R, F-value* graphs for clustering XML documents of synthetic set 2 (MaxRepeats = 10).

Our experimental prototype, including implementations of our XML comparison method and those of [Chawathe 1999], [Nierman and Jagadish 2002] and [Dalamagas *et al.* 2006] is available online for research purposes[1].

5.4. Timing Analysis

Following the complexity analysis developed in Section 4.4, our XML structural similarity method is linear in the number of nodes of each tree, and polynomial (quadratic) in the size of the two trees being compared: $O(|T_1||T_2|)$ (which simplifies to $O(N^2)$, N being the maximum number of nodes in trees T_1 and T_2). This linear dependency on the size of each tree is experimentally verified, timing results being presented in Figure 11. Timing experiments were carried out on a DELL PC with a Xeon 2.66 GHz processor (1GB RAM).

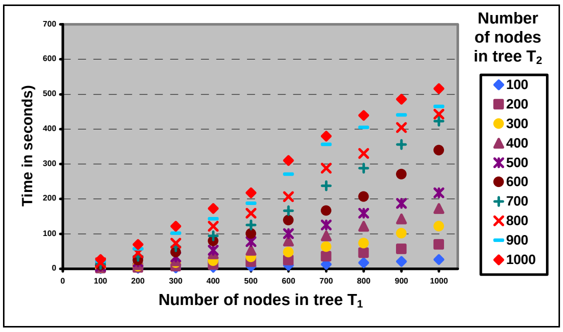


Figure 11. Timing results obtained using our comparison method.

Figure 11 shows that the time to identify the structural similarity between two XML OLTs (Ordered Labeled Trees) of various sizes grows in an almost prefect linear fashion with tree size. Therefore, despite appearing theoretically more complex, timing results demonstrate that our method's complexity, which simplifies to O(|T_1||T_2|), is the same as the approaches in [Chawathe 1999] [Nierman and Jagadish 2002] as well as [Dalamagas *et al.* 2006].

6. Conclusion

In this paper, we proposed a structure based similarity approach for comparing XML documents. Based on a tree edit distance technique, our approach takes into account previously unaddressed sub-tree structural similarities in XML comparison. Our theoretical study as well as

[1] http://www.u-bourgogne.fr/Dbconf/XS2

our experimental evaluation showed that our approach yields improved structural similarity results with respect to existing alternatives, while having the same time complexity (*$O(N^2)$*).

As continuing work, we are exploring the use of our approach in order to compare, not only the structure of XML documents (element/attribute labels) but also their information content (element/attribute values). In such a framework, XML Schemas might have to be integrated in the comparison process, schemas underlining element/attribute data types which are required to compare corresponding element/attribute values. We are also working on extending our approach to encompass semantic similarity assessment between element/attribute node labels while comparing XML documents (taking into account synonyms, antonyms, acronyms, etc., in the edit distance process). In addition, we plan on releasing a public web service version of our experimental prototype.

References

Aho A., Hirschberg D., and Ullman J., Bounds on the Complexity of the Longest Common Subsequence Problem. *Association for Computing Machinery, 23, 1*, 1976, 1-12.

Bertino E., Guerrini G., Mesiti M., A Matching Algorithm for Measuring the Structural Similarity between an XML Documents and a DTD and its Applications, *Elsevier Computer Science, 29*, 2004, 23-46.

Chawathe S., Rajaraman A., Garcia-Molina H., and Widom J., Change Detection in Hierarchically Structured Information. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, Montreal, Quebec, Canada, 1996.

Chawathe S., Comparing Hierarchical Data in External Memory. In *Proceedings of the Twenty-fth Int. Conf. on Very Large Data Bases,* 1999, 90-101.

Cobéna G., Abiteboul S. and Marian A., Detecting Changes in XML Documents. In *Proceedings of the IEEE Int. Conf. on Data Engineering*, 2002, 41-52.

Dalamagas, T., Cheng, T., Winkel, K., and Sellis, T. 2006. A methodology for clustering XML documents by structure. *Inf. Syst. 31, 3,* May. 2006, 187-228.

Flesca S., Manco G., Masciari E., Pontieri L., and Pugliese A., Detecting Structural Similarities Between XML Documents. In *Proc. of the 5th Int. Workshop on The Web and Databases*, 2002.

Gower J. C. and Ross G. J. S., Minimum Spanning Trees and Single Linkage Cluster Analysis, *Applied Statistics*, 18, 1969, 54-64.

Halkidi M., Batistakis Y. and Vazirgiannis M., Clustering Algorithms and Validity Measures, in *SSDBM Conference*, Virginia, USA, 2001.

Levenshtein V., Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Sov. Phys. Dokl., 6*, 1966, 707-710.

Myers E., An O(N D) Difference Algorithm and Its Variations. *Algorithmica*, *1, 2*, 1986, 251-266.

Nierman A. and Jagadish H. V., Evaluating structural similarity in XML documents. In *Proceedings of SIGMOD WebDB'02,* 2002.

Sanz I., Mesiti M., Guerrini G. and Berlanga Lavori R., Approximate Subtree Identification in Heterogeneous XML Documents Collections. *Xsym'05*, 2005, 192-206.

Schlieder T., Similarity Search in XML Data Using Cost-based Query Transformations. In *Proceedings of SIGMOD WebDB'01*, 2001.

Shasha D. and Zhang K., Approximate Tree Pattern Matching. In *Pattern Matching in Strings, Trees and Arrays*, chapter 14, Oxford University Press, 1995.

Wagner J. and Fisher M., The String-to-String correction problem. *Journal of the Association of Computing Machinery*, *21, 1*, 1974, 168-173.

Wong C. and Chandra A., Bounds for the String Editing Problem. *Journal of the Association for Computing Machinery, 23, 1*, January 1976, 13-16.

WWW Consortium, The Document Object Model, http://www.w3.org/DOM.

Zhang K. and Shasha D., Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal of Computing, 18, 6*, 1989, 1245-1262.

Zhang Z., Li R., Cao S. and Zhu Y., Similarity Metric in XML Documents. *Knowledge Management and Experience Management Workshop*, 2003.