

Structure-aware XML Object Identification

Diego Milano, Monica Scannapieco and Tiziana Catarci

Università degli Studi di Roma “La Sapienza”
Dipartimento di Informatica e Sistemistica
Via Salaria 113, 00198 Roma
{milano,monsca,catarci}@dis.uniroma1.it

Abstract

The object identification problem is particularly hard for XML data, due to its structural flexibility. Tree edit distances have been proposed for approximate comparisons among XML trees. However, such distances ignore the semantics implicit in XML data structure, and their use is computationally infeasible for unordered data. In this paper, we define a new distance for XML data, the *structure aware XML distance*, that overcomes these issues, together with a polynomial-time algorithm to calculate it, and we present experimental result that prove its effectiveness and efficiency.

1 Introduction

The *object identification problem* is a central problem arising in data cleaning and data integration, where different objects must be compared to determine if they refer to the same *real-world entity*, even in the presence of errors such as misspellings. As the spread of the XML format as a data model increases, the need to develop effective strategies for XML object identification grows.

XML documents often represent complex, nested data, and schema languages for XML allow great flexibility in how such values are represented inside a document. XML data representations may allow for optional values, and lists of values whose length is not known schema-wise. Functions for approximate XML data comparisons must thus be able to cope both with errors at the level of textual data values and with structural flexibility. The hierarchical nature of XML data has led to the use of *tree edit distances*([?]) to

compare XML documents for various purposes, like detection of differences in versions of XML documents ([?],[?]). Some proposals also address the object identification problem ([?]). Tree edit distances in their original form give great importance to topological features of trees, but are not well suited when node labels and their nesting have semantics and data structure is somewhat regular. Another issue is that, due to the infeasibility of tree edit distance measures for unordered trees ([?]), such proposals are usually based on versions of the tree edit distance for ordered trees. Notice that, while the XML data model is indeed ordered, the presence of unbounded lists of values and optional elements in the data motivates for the adoption of unordered comparisons when looking for approximate matches. As an example, consider an element defined by the following DTD element definition: `<!ELEMENT SHOP (NAME, ADDRESS?, PHONENUM*)>` Here, an object representing a shop may contain zero or more phone numbers. The order in which phone numbers are listed is irrelevant, or however unspecified, so two objects representing the same shop might contain the same set of phone numbers in different order. Requiring that elements correspond to each other in an ordered way may lead to miss some of the similarities among those objects.

We propose a novel distance measure for XML data, the *structure aware XML distance*, that copes with the flexibility which is usual for XML documents, but takes into proper account the semantics implicit in structural information. The structure aware XML distance treats XML data as unordered. Nonetheless, differently from other distances for unordered trees, it can be computed in polynomial time. In this paper, we formally define the structure aware XML distance, we present an algorithm to measure the distance, prove its correctness and its computational cost, and we perform experiments to test the effectiveness and efficiency of our distance measure as a comparison function for XML object identification.

The rest of this paper is organized as follows. In Section 2 we review some related work. In Section 3 we first motivate the introduction of a new distance,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

showing with examples how approaches based on classical tree edit distance fail to respect the semantics of XML data and then define formally the structure aware XML distance. In Section 4 we introduce some theoretical properties of our distance and in Section 5 we present an algorithm to calculate it and we show its correctness and its time complexity. Section 6 describes experimental results. In Section 7 we draw some conclusions and describe some future work.

2 Related Work

The *object identification* problem has been extensively studied for relational data (with the name of record matching or record linkage problem), but the correspondent problem for semi-structured data has only recently drawn some attention. Most proposals for XML object identification are *structure oblivious*, in the sense that they rely on some kind of flattening of document structure to perform comparisons. In [?], XML objects are flattened and compared using string comparison functions. In the DOGMATIX framework([?]), data is extracted from an XML document and stored in relations called *object descriptions*. Tuples of two object descriptors containing data with the same XPath are classified as similar or contradictory using string edit distance, and object descriptions similarity is assessed taking into account the number of similar and contradictory tuples. The approach in [?] is similar, but comparisons of two objects take into account also approximate similarity results of *descendant* objects. *Structure aware* approaches rely on distance measures based on the tree structure of XML, like *tree edit distances* (see [?] and below in this section). In particular, the authors of [?] integrate string comparison functions into the classic tree edit distance for ordered trees to compute approximate joins on XML documents.

The notion of *tree edit distance* for ordered trees is due to Tai ([?]). The problem has also been extended to unordered trees ([?, ?]) and many other variations have been proposed (see e.g. [?, ?, ?]). Most versions of the edit distance problem allow polynomial-time algorithms for the case of ordered trees, but become NP-hard for the unordered case([?]). The *tree alignment distance*([?]) is a restricted version of edit distance. In tree alignment, trees are first made isomorphic (ignoring node labels) with the *insertion* of nodes labelled with *spaces*, and then overlaid. A cost function is defined on pairs of labels and the cost of an alignment is the sum of the costs of opposing labels. An *optimal alignment* is an alignment of minimum cost. Differently from the distance we propose in this paper, tree alignment considers insertions of nodes and overlays nodes with different labels, and it is NP-Hard for unordered trees.

Tree edit distances have been employed also for data and document change detection [?, ?]. The problem

has connections with object identification, but in that context XML documents are mostly modelled as ordered trees, edit operations are extended to entire subtrees, and the focus is on efficiently finding an *edit script* to represent the changes.

3 A Structure-Aware Approach to XML Object Identification

Approaches to solve the object identification problem generally make use of some kind of distance function to detect the similarity of two objects. In record matching techniques proposed for the relational model, attribute values are often compared using *string comparison functions* ([?]). XML documents can be modelled as node labelled trees. This hierarchical, tree-like nature justifies the proposal of similarity measures that integrate string comparison functions with *tree edit distances* ([?]). However, tree distances are not fully able to capture the semantics of XML data, as they do not keep into account the semantics and structural relationships among XML elements.

In this section, we first show some weaknesses that classic tree edit distances suffer when used to compare XML data, and then define a new notion of distance for XML data, the *structure-aware XML distance*, as the basis of an approach to XML object identification.

3.1 Tree Distances

Given a set of edit operations on labelled trees (i.e. node insertions, deletions and relabelling) and a function that assigns a cost to each operation, the *tree edit distance* between two trees is defined ([?]) as the minimum cost sequence of tree edit operations required to transform one tree to another.

Comparison of XML data based on tree distances has been proposed for various purposes ([?, ?, ?]). The authors of [?] perform approximate comparison of XML data with the tree edit distance defined above, using a string comparison function to compute the cost of node relabelling. This approach has the advantage of keeping into account the tree structure of XML data. However, the use of tree edit distance for this purpose has some drawbacks. The examples in Figure 1 illustrate two of them. First, consider the XML data trees **a)** **b)** and **c)**. Tree **c)** represents the same data as tree **a)**, and also contains some additional information. Tree **b)**, instead, represents different data. However, the tree distance between **a)** and **c)** is greater than the distance between **a)** and **b)**. Consider now trees **d)** and **e)**. Here, a person is represented with its parents and an optional list of friends. When measuring the tree edit distance between such two trees, a minimal distance is obtained by deleting from tree **d)** the entire *parent* subtree, relabelling node *friends* into *parents* and matching its leaves to two of the nodes of the *parent* subtree of tree **e)**. This behaviour clearly violates

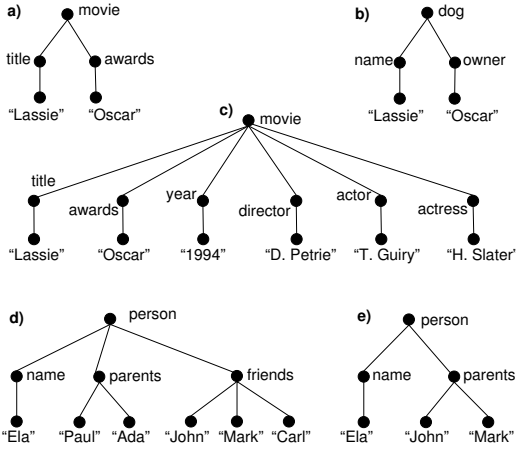


Figure 1: Two issues in classical tree edit distance-based XML comparisons

the semantics implicit in node labels. These problems, in addition to the need of performing unordered comparisons efficiently, motivate the introduction of a new distance measure for XML data.

3.2 XML Structure-aware Distance

In this section, we give an intuitive description of how the issues highlighted in the previous section can be overcome, and then formalize the intuition to define a new distance for XML data. Notice that, throughout this section and in the rest of this paper, we consider XML trees as *unordered*. The above examples show that, when comparing XML trees, a good choice is to match subtrees that have similar structure and that are located under the same path from the root. These can be indeed interpreted as clues of the same semantics. If two trees have exactly the same structure, and only differ by the textual values present on the leaves, we can *overlay* the trees so that nodes with the same path match. When multiple overlays are possible, then we choose one such that the distance among textual values on the leaves is minimal. If the structure of the two trees differ, due to additional information, we can still realize an overlay as above by deleting extra subtrees that do not match well.

The following definitions make the notion of overlay introduce above more formal. We assume a model of XML objects as labelled trees. All leaves are labelled with the same special label τ . Given a leaf l , its textual value (different from its label) is denoted by $text(l)$.

Definition 1 (Overlay) An overlay O of T_1 and T_2 is a non-empty set of pairs of nodes from T_1 and T_2 with the following properties: $\forall v_i, v'_i \in T_i, \forall n_i \in T_i - leaves(T_i), i = 1, 2$,

$$\text{if } \langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle \in O, \text{ then } v_1 = v'_1 \text{ iff } v_2 = v'_2 \quad (1)$$

$$\text{if } \langle v_1, v_2 \rangle \in O, \text{ then } path(v_1) = path(v_2) \quad (2)$$

$$\langle n_1, n_2 \rangle \in O \text{ iff } \exists v_1, v_2 \text{ s.t. } n_1 = parent(v_1) \wedge \quad (3)$$

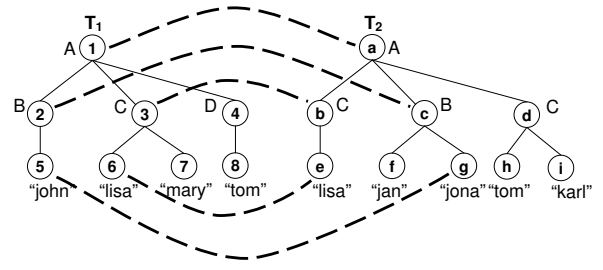


Figure 2: A maximal overlay of two trees

$$n_2 = parent(v_2) \wedge \langle v_1, v_2 \rangle \in O$$

Where $path(v_i)$ denotes the sequence of node labels $label(root_i) \dots label(v_i)$ encountered when traversing T_i from the root to node v_i .

If $\langle v, w \rangle \in O$ we say that v and w *match*. If a node is not matched with any other node, we say that it is *deleted*. Informally, an overlay matches nodes from T_1 to nodes from T_2 one-to-one, so that nodes or leaves are matched only if they have the same path from the root. Two non-leaf nodes can be matched iff they are ancestors of two leaves that are matched. Notice that this implies that, if a node is deleted, all its descendants are also deleted. It also implies that an overlay of two trees exists only if there exist two leaves $l_1 \in T_1$ and $l_2 \in T_2$ with the same path from the root. We say that two trees are *comparable* if they have at least one overlay.

Definition 2 (Maximal overlay) An overlay O of two trees is maximal if there is no other overlay O' such that $O \subset O'$.

Consider the two trees in Figure 2. In the figure, nodes are marked in breadth-first visit order, respectively with numbers and lowercase letters. Uppercase letters beside nodes denote labels, and leaf labels are not shown, while quoted strings denote their textual values. An example of overlay is $O = \{\langle 1, a \rangle, \langle 3, b \rangle, \langle 6, e \rangle\}$. This overlay is not maximal, since there exist other overlays that contain it. An example of maximal overlay that includes O is the overlay shown in the figure using dashed lines $O_c = \{\langle 1, a \rangle, \langle 2, c \rangle, \langle 3, b \rangle, \langle 5, g \rangle, \langle 6, e \rangle\}$. Intuitively, O_c is maximal since it is not possible to add another line from a node of T_1 to a node of T_2 not already touched by a line while maintaining overlay properties. Notice, however, that more than one maximal overlay may exist between two trees. For the trees in Figure 2, another maximal overlay is obtained by matching node 5 with node f rather than node g . Another one is $O'_c = \{\langle 1, a \rangle, \langle 2, c \rangle, \langle 3, d \rangle, \langle 5, g \rangle, \langle 6, i \rangle, \langle 7, h \rangle\}$.

Definition 3 (Cost of a match) Let $sdist(s_1, s_2)$ be a string comparison function. The cost of match for two nodes v, w is:

$$\mu_{v,w} = \begin{cases} sdist(text(v), text(w)) & \text{if } v, w \text{ are leaves} \\ 0 & \text{otherwise} \end{cases}$$

Definition 4 (Cost of an overlay) The cost of an overlay O is defined as $\Gamma_O = \sum_{\langle v,w \rangle \in O} \mu_{v,w}$.

Definition 5 (Optimal overlay) An overlay O of two trees is optimal if it is maximal and there is no other maximal overlay O' such that $\Gamma_{O'} < \Gamma_O$.

Consider again the trees in Figure 2. The cost of the overlay O_c showed in the figure is given by the sum of distances of textual values on matching leaves. Using for instance the common string edit distance ([?]), their distance can be calculated as $sdist(\text{"john"}, \text{"jona"}) + sdist(\text{"lisa"}, \text{"lisa"}) = 2 + 0 = 2$. The cost for overlay O'_c defined above is instead given by $sdist(\text{"john"}, \text{"jona"}) + sdist(\text{"lisa"}, \text{"karl"}) + sdist(\text{"mary"}, \text{"tom"}) = 10$. Actually, O_c is an optimal overlay for the two trees. Notice that more than one optimal overlays may exist for two trees. In this example, the overlay obtained by O_c by matching leaf 5 with leaf f instead of leaf g is still optimal, as the string "john" has the same distance from the strings "jan" and "jona". It is worthwhile to notice that, if two given data trees are comparable, i.e. there is at least an overlay for them, then from the above definitions it follows that there is also a maximal overlay and an optimal overlay for them.

Definition 6 (Structure aware XML distance) The structure aware XML distance of two comparable XML trees T_1 and T_2 is defined as the cost of an optimal overlay of T_1 and T_2 .

Notice that, when applied to the trees in the example given in Figure 1, this distance works as expected. Trees **a)** and **b)** are incomparable, while the distance of trees **a)** and **c)** is zero. In the case of trees **d)** and **e)**, the distance only considers the differences among those leaves that it is meaningful to compare, giving as a result the least distance between names present under the nodes *parents*.

4 Properties of Overlays

In the next section, we present an algorithm to measure the structure aware distance defined in section 3. In this section, we describe some properties of overlays that are useful to prove its correctness. In particular, we show that an optimal overlay of two trees T_1 and T_2 can be found by determining an assignment among the children of their roots such that the sum of the costs of optimal overlays for the corresponding subtrees is minimal (Theorem 4). To prove this result, we first show that the cost of an overlay of two trees can be rewritten in terms of the cost of overlays of the children of their roots (Theorems 1 and 2).

For space reasons, we omit the proofs of some theorems and only sketch other proofs. Throughout this section, we denote with T_1, T_2 two comparable data trees, with r_1, r_2 their roots, and with

$v_i, w_j, i \in [1, deg(r_1)], j \in [1, deg(r_2)]$ the children of r_1 and r_2 , respectively. Furthermore, given a node v , we denote with $T(v)$ the tree rooted at v . We call the trees $T(v_i)$ and $T(w_j)$ the *first-level subtrees* of T_1 and T_2 respectively.

Theorem 1 Let O be an overlay of T_1, T_2 . If $\langle v, w \rangle \in O$, then the set $O_{v,w} = \{\langle y, z \rangle \in O \mid y \in T(v), z \in T(w)\}$ is an overlay of $T(v)$ and $T(w)$. Moreover, if O is maximal then also $O_{v,w}$ is maximal. We say that $O_{v,w}$ is the overlay induced by O on $T(v)$ and $T(w)$.

Proof sketch: To show that $O_{v,w}$ is an overlay of $T(v)$ and $T(w)$, we show that properties (1),(2) and (3) of overlays are respected. The fact that $O_{v,w}$ is also maximal can be proved by contradiction. If there exist another overlay of $T(v)$ and $T(w)$ $O'_{v,w} \supset O_{v,w}$, then there exists another overlay $O' \supset O$, and thus O is not maximal. ■

Theorem 2 Let O be an overlay of two trees T_1, T_2 . Then

$$O = \langle r_1, r_2 \rangle \cup \left(\bigcup_{\langle v_i, w_j \rangle \in O} O_{v_i, w_j} \right) \quad (4)$$

$$\forall \langle v_i, w_j \rangle, \langle v_h, w_k \rangle \in O, O_{v_i, w_j} \cap O_{v_h, w_k} = \emptyset \quad (5)$$

Proof sketch: (4) By definition, $O_{v_i, w_j} \subset O$ and thus $\langle r_1, r_2 \rangle \cup \left(\bigcup_{\langle v_i, w_j \rangle \in O} O_{v_i, w_j} \right) \subset O$. It remains to show that the reverse inclusion holds, i.e. that for any match $\langle y, z \rangle \in O$, $\langle y, z \rangle \in \{\langle r_1, r_2 \rangle\} \cup \left(\bigcup_{\langle v_i, w_j \rangle \in O} O_{v_i, w_j} \right)$. This can be shown reasoning on the depth of $\langle y, z \rangle$. (5) can be proved showing that for any two overlays $O_{v_i, w_j}, O_{v_h, w_k}$, if their intersection is not empty, then necessarily $v_i = v_h$ and $w_j = w_k$. ■

In other words, an overlay of T_1 and T_2 is a partition of the overlays it induces on its first-level subtrees and the couple $\langle r_1, r_2 \rangle$.

Theorem 3 The cost of a maximal overlay O is the sum of the costs of all the overlays O_{v_i, w_j} induced on its first level subtrees.

Proof: Follows immediately from Theorem 2. ■ From this result it follows, trivially, that an overlay is optimal only if the overlays induced on its first level subtrees are all optimal. Notice that the reverse does not hold in general. As an example, consider the overlay shown in Figure 4. It is easy to see that the overlays $O_{3,b}$ and $O_{4,c}$ are both optimal since $sdist(\text{"john"}, \text{"joe"}) + sdist(\text{"mary"}, \text{"mark"}) < sdist(\text{"john"}, \text{"mark"}) + sdist(\text{"mary"}, \text{"joe"})$. However, the overlay of T_1 and T_2 shown in the picture is not optimal, since another overlay with cost 0 can be obtained by matching node 3 with c and node 4 with b. In order to reach a necessary and sufficient condition for the optimality of an overlay, we introduce a few more definitions. Given two trees T_1 and T_2 , a *first-level assignment* of T_1 and T_2 is a set of couples

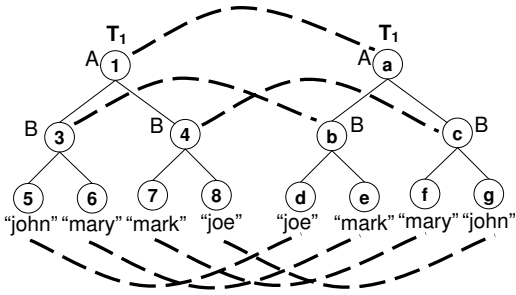


Figure 3: The overlay of T_1 and T_2 is not optimal, even if all the overlays induced on subtrees are optimal

$\langle v_i, w_j \rangle$ such that each node of each tree is coupled with at most another node of the other one, and the trees $T(v_i)$ and $T(v_j)$ are comparable. The concept of maximality defined for overlays can be easily extended to first level assignments. It is easy to prove that an overlay is maximal iff includes a maximal assignment. By defining the *cost* of a maximal first-level assignment as the sum of the costs of the overlays of first level trees whose roots are coupled in the assignment, the concept of optimality can also be extended to first-level assignments. Given this definition, we can now present the main result of this section:

Theorem 4 *An overlay O is optimal iff it contains an optimal first-level assignment A and the overlays induced on its first level subtrees are all optimal.*

Proof sketch: Maximality follows from the maximality of A . From the optimality of A and of the overlays induced of first level subtrees it follows, by Theorem 3, that the overlay is also optimal. ■

The previous result gives an operative way to build an optimal overlay of two trees T_1 and T_2 . A maximal assignment can be obtained by first matching r_1 with r_2 and then matching a children of r_1 with a children of r_2 until no more matches are possible. Two nodes v_i and w_j are matched only if $T(v_i)$ and $T(v_j)$ are comparable. In this case, an optimal overlay is built for $T(v_i)$ and $T(v_j)$ by applying the same process, recursively, up to the leaves. All possible maximal assignments must be built and evaluated, and an optimal one must be chosen.

5 Structure Aware XML Distance Measurement

In this section, we introduce an algorithm to measure the structure aware XML distance defined in Section 3, and prove its correctness and its worst case complexity.

Algorithm 1 analyzes two comparable trees recursively, starting from the roots. If the roots are leaf nodes, a distance measure for their associated text values is returned. Such function is denoted by the procedure *sdist()* in the algorithm. Otherwise, the algorithm considers their children, and computes a distance for each couple of subtrees rooted at children

Algorithm 1 $dist(T_1, T_2)$

```

if isLeaf( $r_1$ ) and isLeaf( $r_2$ ) then
  return sdist(text( $r_1$ ), text( $r_2$ ))
else
   $xmldist := \infty$ 
  for all  $l$  in labels(children( $r_1$ )  $\cup$  children( $r_2$ )) do
    for all  $v_i \in \textit{children}_l(r_1)$  do
      for all  $w_j \in \textit{children}_l(r_2)$  do
         $D_l[i, j] := \textit{dist}(T(v_i), T(w_j))$ 
      end for
    end for
     $assignment_l := \textit{findAssignment}(D_l[])$ 
    for all  $\langle h, k \rangle \in assignment_l$  do
      if  $xmldist = \infty$  then
         $xmldist := 0$ 
      end if
       $xmldist := xmldist + D_l[h, k]$ 
    end for
  end for
  return xmldist
end if

```

with the same label, recursively. After all distances have been calculated, the algorithm must assign each node to another node with the same label, minimizing the overall cost. This is an assignment problem and can be solved using a variation of the well-known Hungarian Algorithm ([?]). In the algorithm, this task is performed by a call to procedure *findAssignment()*. In particular, given a matrix of distances, the procedure returns a set of assignments containing couples of indices of assigned nodes. For ease of presentation, in the algorithm we denote the set of all children of node v having label l with $\textit{children}_l(v)$. Results of distance calculations for a certain set of children having label l are stored in an array named D_l . The distance is initially set to ∞ , and reset to 0 only in the case that there is at least one assignment of root children.

From the results given in the previous section and the definition of structure aware XML distance given in section 3 it follows immediately that:

Theorem 5 *Algorithm 1 correctly computes the structure aware XML distance of two comparable trees.*

In order to understand the computational cost of the algorithm, let us consider a case in which all the leaves of the tree have the same path, and the data trees are maximal. We consider distance calculation among two trees T_1 and T_2 . We denote with deg_1 and deg_2 their respective degrees and with L_1, L_2 their sets of leaves.

Let T'_1, T'_2 be two subtrees of T_1 and T_2 rooted at level l , and let r'_1, r'_2 be their roots. In order to compute their distance, we must choose a match among the children of r'_1 and r'_2 such that the the sum of distances for corresponding subtrees is minimal. Assuming that we have already calculated all pairwise distances, we need to solve an instance of the linear assignment problem. The Hungarian algorithm gives a solution in cubic time, so the cost of an assignment is $O((deg_1 + deg_2)^3)$.

To compute all distance measurements, we proceed bottom up, starting from the leaves and calcu-

lating all pairwise distances among all nodes at each level. At level $depth - 1$, before performing the assignment phase we must compute distances among textual values. These are computed in constant time (w.r.t. the size of the trees). At upper levels, we already know the distances among nodes at lower levels, so we just need to perform the assignment phase. In total, the assignment phase is repeated $(|T_1| - |L_1|) \times (|T_2| - |L_2|)$ times. Thus, the overall cost is $O((|T_1| - |L_1|) \times (|T_2| - |L_2|) \times (deg_1 + deg_2)^3)$. When there is more than one path for leaf nodes, the calculation is less expensive.

6 Experiments

This section presents an experimental evaluation of the structure aware XML distance introduced in the previous sections. We tested both *effectiveness* and *efficiency* of our distance as the basis of a comparison function for XML Object identification.

6.1 Experimental Setting

In the tests, a set of objects contained in an XML document are compared pairwise for similarity. Two objects o_i, o_j are classified as similar if $dist(o_i, o_j) < t$, where $dist$ is the structure aware XML distance and t is a fixed threshold. The string comparison function we use is the string-edit distance([?]). We performed three sets of tests. Tests in the first and second set aim at determining how good is our measure at correctly identifying similar objects under various experimental conditions. The third set was performed to compare our distance with the tree-edit distance used in [?].

6.2 Data Sets

The experiments were performed on a synthetic data set created with ToXGene¹. The objects compared for similarity represented persons, defined as in the following DTD element declarations:

```
<ELEMENT PERSON (NAME, MIDDLENAME*, SURNAME, ADDRESS)>
<ELEMENT ADDRESS (STREET, CITY, COUNTRY, EMAIL*, PHONENUM*)>
```

all elements not defined above contained PCDATA. Textual data values were taken from the XMark benchmark; *middlename*, *email* and *phonenumber* optional elements were limited to a maximum of respectively to 2, 2, and 4 instances for each person. We started from a clean data set of 300 distinct *person* objects. In all experiments, for each *person* object another similar object was created (duplication rate = 100%). In order to create similar objects under controlled conditions, we have developed a small tool² that allows to produce objects that differ from the original by a given rate of *internal data deletions*, *text changes*, and *internal data duplications*, and *element swaps*. Internal data deletions are leaf deletion. By internal data duplication

we mean that a portion of an object (e.g. an *middlename* element or an *address* subtree) is duplicated *inside the object*. These internal duplicates receive the same amount of changes as other duplicates, except for internal duplication. Text changes consisted of deletions, insertions and character swaps. We inserted a minimum of 2 errors for each text modification. The specific kind of errors were chosen randomly with equal probability (e.g. a character deletion and a swap, or two swaps).

Notice that change rates mentioned in the tests refer to the rate of changes *for each duplicate*. That is, all duplicates differ from their original, and the change rate refers to which percentage of the data present in a duplicated object has received such changes. As an example, a deletion rate of 20% means that in the duplicate of a certain object 20% of the leaves were deleted. A text change of 10% means that 10% of the textual values present on leaves that were not deleted are altered with a certain number of deletions, insertions and character swaps.

6.3 Experimental Set 1

A first set of experiments tested how the effectiveness of our measure varies depending on the chosen threshold. Effectiveness is measured in terms of the f-score:

$$Fscore = 2 \cdot recall \cdot precision / (recall + precision)$$

Where *recall* is the percentage of matches identified by the algorithm, and *precision* is the percentage of actual matches among those declared by the algorithm.

We let the threshold vary over a wide range of values, and performed the test for files in which the differences among duplicate object are set as follows:

case	deletion rate	text change rate
1	10%	10%
2	10%	20%
3	20%	20%
4	20%	30%

Observe that the last one is a rather critical situation: a duplicate for an object with 10 leaves has 2 leaves removed and three of the remaining leaves contain errors. The results for this set of experiments are shown in Figure 4. A first observation that must be done is that in all cases the measure achieves an f-score of 100% for some threshold values. Thus, the distance shows high effectiveness as an object identification comparison function. It is also worthwhile to notice that all the curves in Figure 4 show a wide plateau in which the f-score stay at its maximum value. Since, in real-use conditions, determining the right threshold is mainly a matter of experience and it is often infeasible to evaluate various thresholds, the relative insensibility of the distance to threshold variation in a wide range of threshold must be considered as a positive feature. The curves in the graph appear grouped

¹www.cs.toronto.edu/tox/toxgene

²www.dis.uniroma1.it/milano/duplicatcreator

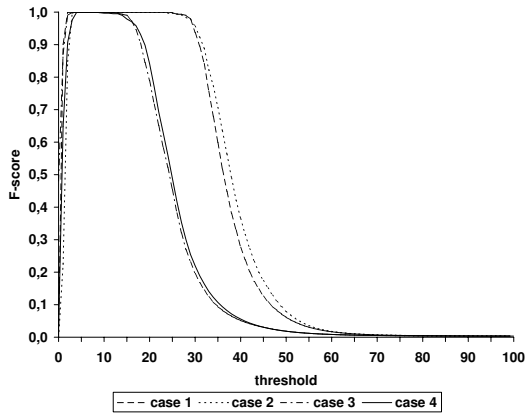


Figure 4: Results for experimental set 1: fscore dependency on similarity threshold

in two sets, corresponding to the two values of deletion rate used in the experiment. This behavior means that the distance is more sensible to a variation in the data available for comparison than to diffused errors over the data itself.

6.4 Experimental Set 2

Our second set of experiments was performed to determine how our distance is influenced by a variation in the kind and amount of differences among similar objects. The threshold was fixed to a value of 10, and the rate differences introduced in duplicate objects were varied independently. Figures 6(b), 6(a) and 6(c) show respectively the trend of the recall, precision and f-score measures for these tests. Each graph contains different curves, each one relative to a specific kind of differences. Different kinds of changes affect in different way the behavior of the measure. From the graphs it appears that a high rate of deletions affects precision but not recall. This is due to the fact that deletions reduce the amount of comparable information available to determine if two objects are similar. On the other hand, high text change rates mainly influence recall, since a high rate of errors may introduce high differences in comparable features of objects. The measure is completely insensible to swaps of elements. This is not surprising, since it compares unordered trees.

6.5 Experimental Set 3

We described in section 3 some issues related to ordered tree edit distances when used in an object identification context, and we claimed that our distance is both more efficient and effective than such class of distances. Our third set of experiments confirms this claim. We tested our distance measure and the ordered tree-edit distance over the same set of files, and compared both execution time and f-score. The algorithm used for the ordered tree edit distance is described in [?], and the cost function was modified to account for

textual value comparison as proposed in [?]. More specifically, the cost for relabelling, insertion and deletion for leaves was evaluated based on the same string comparison function used in our measure. For internal nodes, a unit cost function was used. The files used in the tests contained duplicates produced by applying to each object all the four kinds of changes described in section 6.2, with the same change rate for each difference. Thus, a *data change* rate of 10% means that each duplicate has data deletion, text changes, internal duplication, and element swap rates all set to 10%. We let the change rate vary from 10% to 45% at intervals of 5%. The threshold was fixed. The graphs shown in Figure 5 refer to the results obtained for the best possible choice of a threshold for both measures. The differences in execution time for the two distances were dramatic. To perform the comparison over a total of 600 objects the ordered tree edit distance took approximately 7,5 hours while, on the same machine and for the same data, our distance takes something more than a minute. The results shown in Figure 5 highlight how our distance constantly outperforms the ordered tree edit distance from the effectiveness point of view. This differences are partly due to the fact that our distance performs an unordered comparison, and the files contain swapped elements. Notice that only adjacent elements with the same name where swapped in the test files, thus respecting the DTD of the data. Another difference is also due to the fact that our measure ignores internally duplicate data that cannot be matched, while the tree edit distance accounts for the cost of deletion of such data.

7 Conclusions

XML data has tree-like nature and flexible structure. These features have lead to proposals for XML object identification that exploit tree-edit distances to perform approximate comparisons among XML trees.

The tree-like nature and flexible structure of XML

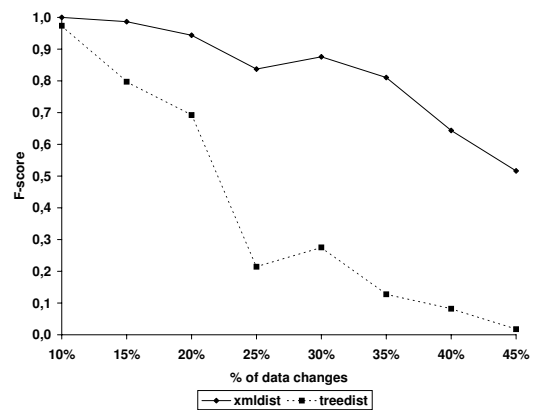


Figure 5: Results for experimental set 3: comparison between tree edit distance and structure aware XML distance

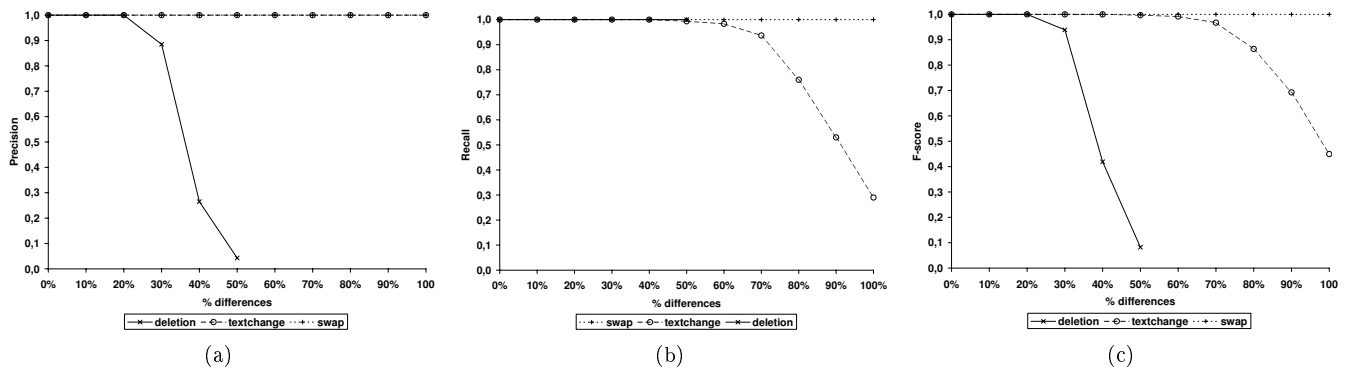


Figure 6: Precision, recall and f-score for experimental set 2

data are serious issues in XML object Identification. Such features have lead to proposals that exploit tree-edit distances to perform approximate comparisons of XML trees. However, tree-edit distances ignore the semantics implicit in element labels and nesting relationships. Furthermore, while tree-distances for unordered trees are better suited to perform approximate comparisons of XML data, their use is computationally infeasible. In this paper, we have defined a new distance for XML data, the *structure aware XML distance*, that overcomes these issues. The distance compares only portions of XML data trees whose structure suggest similar semantics. Furthermore, it performs comparison on unordered trees, without incurring in high computational costs. We have presented an algorithm to measure the distance between two trees, and discussed its complexity, that is polynomial. We also presented an experimental evaluation of our measure as the basis of an object identification approach.

Our measure is suited for detecting similar objects when the scheme of objects is approximately the same, as in the case of a single data source, or several data sources on which a schema integration activity has already been performed. We are investigating how to add more flexibility without sacrificing the gain in efficiency we have obtained. We also plan to investigate other issues related to the use of tree distances for XML comparison. In [?], the authors suggest the use of ontology-based techniques to evaluate the cost of relabelling nodes. How to balance the effects of string-comparison-based and ontology-based cost evaluation seems far from trivial.