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, monscan, catarci}@dis.uniroma1.it

† Istituto Nazionale di Statistica
Via Cesare Balbo 6, 00184 Roma
scannapi@istat.it

Abstract

The object identification problem is particularly difficult for XML data, due to its structural flexibility. Tree edit distances have been used to perform approximate comparisons among XML trees. However, such distances ignore the semantics implicit in element labels and nesting relationships of XML data. Furthermore, the use of tree edit distances for unordered trees, that would be more suitable for this task, is computationally infeasible. In this paper, we define a new distance for XML data, the structure aware XML distance*, that overcomes these issues, and present a polynomial algorithm to calculate it.*

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. As a data model, XML is half a way between completely semistructured models, in which nothing is known in advance about the structure of the data, and structured ones, like the relational model. XML documents often represent complex, nested data. However, they are usually required to conform to some kind of structural specification, expressed in schema languages like DTD and XML Schema. Hence, their structure, though flexible, usually exhibits a certain degree of regularity.

Flexibility is one of the major issues in XML object identification. 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 this structural flexibility. The hierarchical nature of XML data has lead to the use of *tree edit distances*([1]) to compare XML documents for various purposes, like approximate DTD matching and detection of differences in versions of XML documents ([3]). Some proposals also address the object identification problem ([4]).

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. The proposals cited above adapt or extend tree edit distances to handle XML documents, but ignore this problem.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Another issue is that, due to the infeasibility of tree edit distance measures for unordered trees([18]), such proposals are usually based on versions of 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 believe that unordered comparison are much more suited than ordered comparisons to perform approximate matches in data-oriented XML.

We are currently investigating how to overcome these issues. This paper presents the first results of our ongoing research. In particular, we propose a novel distance measure for XML data, the *structure aware XML distance*, and present an algorithm to compute it. This distance copes with the flexibility which is usual for XML documents, but takes into proper account the semantics implicit in structural information. It allows for comparison of XML data as unordered tree, and is thus particularly suited to identify objects that may differ also for the order of the data values they contain. Nonetheless, differently from other distances for unordered trees, it can be computed in polynomial time. The structure aware XML distance can be used as the basis for approximate comparisons in an XML object identification framework.

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, showing with examples how approaches based on classical tree edit distance fail to respect the semantics of XML data. We then define formally the structure aware XML distance. In Section 4 we present an algorithm to calculate the distance on two XML trees and review its time complexity. In Section 5 we draw some conclusions, and describe some issues we plan to consider in our 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 [16], XML objects are flattened and compared using string comparison functions. In the DOGMATIX framework([15]), 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 [13] is similar, but objects have, beside flat object descriptors, also nested objects called *descendants*. Comparisons among objects are performed level by level in a bottom up fashion, and approximate similarity results at lower-levels are considered when comparing objects at upper levels. In particular, the similarity of two objects is influenced by how many of their descendants are supposed to be similar. Notice that, in contrast to this approach, the distance we define in this paper takes into account the entire structure of an XML tree at once. *Structure aware* approaches proposed for XML object identification rely on distance measures based on the tree structure of XML, like *tree edit distances* (see [1] and below in this section). In particular, the authors of [4] integrate string comparison functions into the classic tree edit distance for ordered trees to compute approximate joins on XML documents. The paper is mainly concerned with heuristics to filter unneeded comparisons, based on efficient computation of bounds for the distance.

It is worthwhile to notice that tree edit distances have been used to measure similarity of XML documents also in proposals not directly related to object identification. As an example, in [3] the authors use a version of tree edit distance that allows moving entire subtrees, in order to detect changes in hierarchically structured documents. In [9], an analogous version of edit distance is used to cluster XML documents by similarity.

The notion of *tree edit distance* for ordered trees has been introduced in its most widely known form by Tai ([14]), though a restricted version already appeared in [11]. Since then, the problem has been extended to unordered trees ([18, 12]) and many variations have been proposed (see e.g. [5, 17, 10]). Most versions of the edit distance problem allow polynomial-time algorithms in the case of ordered trees, but become NP-hard in the unordered case([18]). The *tree alignment distance*([5]) 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 overlayed. 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. The alignment problem has polynomial cost for ordered trees, but becomes NP-Hard for unordered trees. In [10] a *structure respecting edit distance* is presented. Despite the similarity with the name of the distance introduced here, the proposal in [10] is a tree edit distance, with the added constraint that disjoint subtrees are mapped to disjoint subtrees, and does not take into account the semantics of structure, as in our case. We refer to [1] the reader interested in a survey on various versions of tree edit distances.

3 A Structure-Aware Approach to XML Object Identification

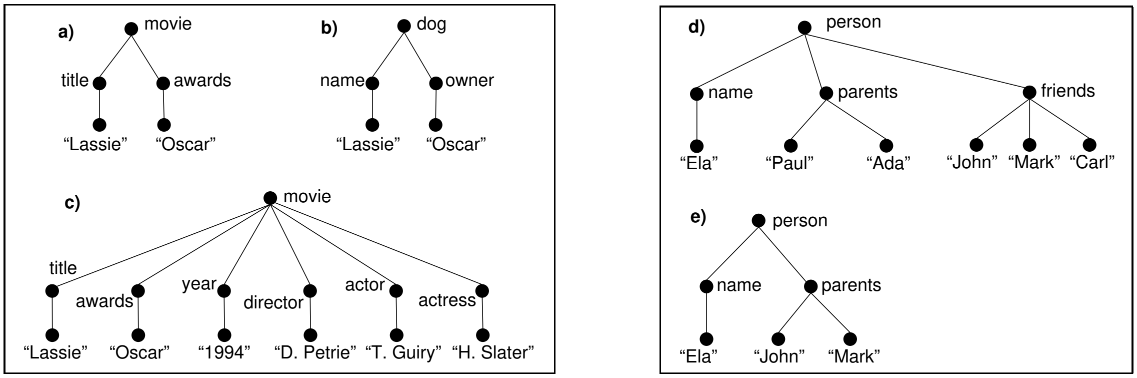
Approaches to solve the object identification problem generally make use of some kind of distance function to detect the similarity of two objects. In record matching techniques proposed for the relational model, attribute values are often compared using *string comparison functions* ([8, 6]). XML documents 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* ([1]). Tree distances have been introduced to measure structural similarity among trees. However, they 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 relabellings) and a function that assigns a cost to each operation, the *tree edit distance* between two trees is defined ([14]) as the minimum cost sequence of tree edit operations required to transform one tree to another. Other variants of this notion have been proposed in the literature, including versions where edit operations are allowed on entire subtrees.

Comparison of XML data based on tree distances has been proposed for various purposes ([4, 3, 9]). As an example, in [4] the authors adapt the tree distance defined above to perform approximate XML joins by adding comparisons of text node labels based on a string comparison function. With respect to other proposals for XML object identification, 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 following examples illustrate two of them. First, consider the XML data trees represented in Figure 1(a). Tree **c)** represents the same data as tree **a)**, and also contains some additional information. Tree **b)**, instead, represents different data. However, as it can be easily verified, the tree distance between **a)** and **c)** is greater than the distance between **a)** and **b)**. Let us now consider the example shown in Figure 1(b). Here, a person can be represented with its



(a) (b)

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

parents and an optional list of friends. When measuring the distance between the two trees **d)** and **e)**, 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 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 first give an intuitive description of our approach to distance measurement, and then we formalize the distance itself. Our aim is to overcome the problems highlighted in the previous section by taking into full account the presence of structural information in XML data.

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 two trees have different structure, due to the presence of additional information, we can match those subtrees that exhibit the least distance on leaves, and ignore those that cannot be matched. Intuitively, we are trying to realize an overlay as above by deleting extra subtrees that do not match well. In the remainder of this section we formalize these intuitions, and use them to define a new distance for XML data.

An *overlay* O of two data trees T_1 and T_2 is a non-empty set of couples of nodes from T_1 and T_2 with the following properties. Let $v_i, v_i' \in T_i, n_i \in (T_i - leaves(T_i)), i = 1, 2$:

$$\begin{cases} \text{if } \langle v_1, v_2\rangle, \langle v_1', v_2'\rangle \in O, \text{ then } v_1 = v_1' \text{ iff } v_2 = v_2'; \\ \text{if } \langle v_1, v_2\rangle \in O, \text{ then } path(v_1) = path(v_2); \\ \langle n_1, n_2\rangle \in O \text{ iff } \exists v_1, v_2 \text{ s.t. } n_1 = parent(v_1) \wedge n_2 = parent(v_2) \wedge \langle v_1, v_2\rangle \in O. \end{cases}$$

Where, $path(v)$ denotes the sequence of node labels $label(root) \ldots label(v)$ encountered when traversing the tree from the root to node v. If $\langle v_1, v_2\rangle \in O$ 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 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. An overlay O of two trees is *complete* if there is no other overlay O' such that $O \subset O'$. In the rest of this paper, when we refer to an overlay, we implicitly assume that it is a complete overlay.

Let $sdist(s_1, s_2)$ be a string comparison function. If v is a leaf node, we denote with $text(v)$ its string value. The *cost of match* for two nodes v, w is:

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

The *cost of an overlay O* is defined as $\Gamma_O = \sum_{\langle v,w \rangle \in O} \mu_{v,w}$. An overlay O of two trees is *optimal* if it is complete and there is no other complete overlay O' such that $\Gamma_{O'} < \Gamma_O$.

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. In Section 4, we describe an algorithm that measures the structure aware XML distance among two trees. Differently from other tree-distances, the distance defined here can be computed in polynomial time, even if the trees are unordered. Therefore, this distance is suitable for object identification, which requires a high number of pairwise comparisons among trees.

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

4 Structure Aware XML Distance Measurement

In this section, we introduce an algorithm to measure the structure aware XML distance defined in Section 3. Before presenting it in details, we describe some properties of overlays that are useful to understand how the algorithm works. 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.

Let O be an overlay of T_1, T_2. It can be easily shown that if $\langle v, w \rangle \in O$, then the set $O_{v,w} = \{\langle y, z \rangle \in O | y \in T(v), z \in T(w)\}$ is an overlay of $T(v)$ and $T(w)$. Therefore, O can be written $O = \langle r_1, r_2 \rangle \cup (\bigcup_{\langle v_i, w_j \rangle \in O} O_{v_i,w_j})$. It follows that the cost of O is the sum of the costs of all the overlays O_{v_i,w_j}. It is also immediate to see that O is optimal only if the overlays O_{v_i,w_j} are all optimal.

A complete overlay of T_1 and T_2 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. Notice that nodes can be matched only if they have the same label. Nodes that are not matched are deleted, along with all their descendants. If v_i and w_j are matched, then an overlay is built for $T(v_i)$ and $T(v_j)$ by applying the same process, recursively, up to the leaves. From the above considerations, it follows that, in order to obtain an optimal overlay, the children of r_1 and r_2 must be matched so that the sum of the costs of optimal overlays for subtrees rooted at matched nodes is minimal. In other words, an algorithm must choose, among all possible assignments of subtrees rooted at the children of r_1 to subtrees rooted at the children of r_2, one that minimizes the sum of the distances of all couples of subtrees.

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 $compareStrings()$ in the algorithm. Otherwise, the algorithm considers their children, and computes a distance for each couple of subtrees rooted at children 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 ([7],[2]). 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

Algorithm 1 $StructureAwareXMLDist(T_1, T_2)$

```
if isLeaf(r_1) and isLeaf(r_2) then
  returncompareStrings(text(r_1), text(r_2))
else
  structDist := ∞
  for all l in labels(children(r1) ∪ children(r_2)) do
    for all v_i ∈ children_l(r_1) do
      for all w_j ∈ children_l(r_2) do
        children_lDistance[i, j] := StructureAwareXMLDist(T(v_i), T(w_j))
      end for
    end for
    assignment_l := findAssignment(children_lDistance[])
    for all ⟨h, k⟩ ∈ assignment_l do
      if structDist = ∞ then
        structDist := 0
      end if
      structDist := structDist + children_lDistance[h, k]
    end for
  end for
  return structDist
end if
```

nodes. For ease of presentation, in the algorithm we denote the set of all children of node v having label l with $children_l(v)$. Results of distance calculations for a certain set of children having label l are stored in an array named $children_lDistance$. The distance is initially set to ∞, and reset to 0 only in the case that there is at least one assignment of root children.

In order to understand the 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 complete. 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 calculating 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. For example, if the sets of children of two nodes are partitioned according to their label in two sets of equal cardinality s, in order to compute the distance among the two nodes the algorithm will have to calculate $(s^2 + s^2)$ distances among children, instead than $(s + s)^2$, and calculate two assignments at the cost of $O((2s)^3)$ instead of a single one at cost $O((4s)^6)$.

5 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. However, tree edit distances suffer from certain drawbacks, since they ignore the semantics implicit in the 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.

In our future work, we will perform experiments to determine the effectiveness of our distance as the basis of an object identification approach. We also plan to investigate other interesting issues related to the use of tree distances for XML comparison. As an example, in [4] the authors suggest the use of ontology based techniques to evaluate the cost of relabelling element nodes. How to balance the effects of string- comparison-based and ontology- based cost evaluation seems far from trivial.

References

[1] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3), 2005.

[2] Francois Bourgeois and Jean-Claude Lassalle. An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Commun. ACM*, 14(12):802–804, 1971.

[3] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6*, 1996.

[4] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate xml joins. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6*, 2002.

[5] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees - an alternative to tree edit. *Theor. Comput. Sci.*, 143(1):137–148, 1995.

[6] Nick Koudas and Divesh Srivastava. Approximate joins: Concepts and techniques. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2*, 2005.

[7] James Munkres. Algorithms for assignment and transportation problems. *SIAM Journal on Computing*, 5(1), March 1957.

[8] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[9] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *Proceedings of the Fifth International Workshop on the Web and Databases, WebDB 2002, Madison, Wisconsin, USA, June 6-7, 2002, in conjunction with ACM PODS/SIGMOD 2002*, pages 61–66, 2002.

[10] Thorsten Richter. A new measure of the distance between ordered trees and its applications. Technical Report 85166-CS, 1997.

[11] Stanley M. Selkow. The tree-to-tree editing problem. *Inf. Process. Lett.*, 6(6):184–186, 1977.

[12] Dennis Shasha, Jason Tsong-Li Wang, Kaizhong Zhang, and Frank Y. Shih. Exact and approximate algorithms for unordered tree matching. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4), 1994.

[13] Felix Naumann Sven Puhlmann, Melanie Weis. Xml duplicate detection using sorted neighborhoods. In *Proceedings of EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31*. Springer Verlag, LNCS 3896, 2006.

[14] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3), 1979.

[15] Melanie Weis and Felix Naumann. Dogmatix tracks down duplicates in xml. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*.

[16] Melanie Weis and Felix Naumann. Detecting duplicate objects in xml documents. In *IQIS 2004, International Workshop on Information Quality in Information Systems,Paris, France*, 2004.

[17] Kaizhong Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3), 1996.

[18] Kaizhong Zhang, Richard Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3), 1992.