Structural Similarity Evaluation between XML Documents and DTDs

Joe Tekli[1], Richard Chbeir[1] and 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. The automatic processing and management of XML-based data are ever more popular research issues due to the increasing abundant use of XML, especially on the Web. Nonetheless, several operations based on the structure of XML data have not yet received strong attention. Among these is the process of matching XML documents and XML grammars, useful in various applications such as documents classification, retrieval and selective dissemination of information. In this paper, we propose an algorithm for measuring the structural similarity between an XML document and a Document Type Definition or DTD considered as the simplest way for specifying structural constraints on XML documents. We consider the various DTD operators that designate constraints on the existence, repeatability and alternativeness of XML elements/attributes. Our approach is based on the concept of tree edit distance, as an effective and efficient means for comparing tree structures, XML documents and DTDs being modeled as ordered labeled trees. It is of polynomial complexity, in comparison with existing exponential algorithms. Classification experiments, conducted on large sets of real and synthetic XML documents, underline our approach's effectiveness, as well as its applicability to large XML repositories and databases.

Keywords: Semi-structured XML-based data, XML grammar, DTD, structural similarity, tree edit distance.

1 Introduction

Computing similarity between the ever-increasing number of XML documents is becoming essential in several scenarios such as in change management and data warehousing [7], [8], [9], XML query systems (finding and ranking results according to their similarity) [28], [36] as well as the clustering of similar XML documents gathered from the web [10], [24]. Similarly, estimating similarity between XML grammars is useful for data integration purposes, in particular the integration of Document Type Definitions/schemas that contain nearly or exactly the same information but are constructed using different structures [12], [13], [21], [22].

On the other hand, evaluating similarity between documents and grammars can also be exploited in various domains, e.g. for classifying XML documents against a set of DTDs/schemas declared in an XML database (just as DB schemas are necessary in traditional DBMS for the provision of efficient storage, retrieval, protection and indexing facilities, the same is true for DTDs and XML repositories) [3], [4], [24], XML document retrieval via structural queries [4], [15] (a structural query being represented as a DTD in which some additional constraints on data content could be posed), as well as in the selective dissemination of XML documents (user profiles being expressed as DTDs against which the incoming XML data stream is matched. Hence, an XML document would be distributed to the users whose profiles – DTDs – are similar enough to the document) [3], [4].

In this study, we focus on the *document/grammar* comparison issue, introducing a novel method for evaluating the structural similarity between an XML document and a DTD. Different from previous heuristic approaches, e.g. [3], [4], our method makes use of the well know techniques for finding the edit distance between tree structures, XML documents and DTDs being modeled as ordered labeled trees (OLTs). The contributions of the paper can be summarized as follows: i) introducing a tree representation model, that copes with the

expressive power of XML DTD grammars, ii) introducing a polynomial tree edit distance algorithm for evaluating the structural similarity between an XML document and a DTD, iii) developing a prototype to evaluate and validate our approach.

The rest of the paper is organized as follows. Section 2 reviews background in XML/grammar similarity approaches and related problems, such as approximate pattern matching and tree edit distance. Section 3 presents preliminary definitions and provides the tree representations of XML documents and DTDs adopted in this study. In Section 4, we develop our XML/DTD structural similarity approach. Section 5 presents our experimental tests. Section 6 concludes the paper and outlines some future research directions.

2 Related Work

In the following, we briefly review the problem of approximate pattern matching with *variable length don't cares (VLDC)*, which is seemingly comparable to the issue of XML/DTD matching. Then, we present a glimpse on the state of the art concerning tree edit distance algorithms, widely employed to compare semi-structured data, XML documents in particular. Subsequently, we go over existing XML/Grammar similarity computing algorithms.

2.1 Approximate Pattern Matching with Variable Length Don't Cares

An intuitive XML/DTD comparison approach could be that of approximate matching with the presence of *Variable Length Don't Cares (VLDC)*. In strings, a *VLDC* symbol (e.g. $\wedge$) in a given pattern may substitute for zero or more symbols in the data string [2] [17]. Approximate *VLDC* string matching means that after the best possible substitutions, the pattern still does not match the data string and thus a matching distance is computed. For example, "*comp* $\wedge$ *ng*" matches "*commuting*" with distance 1 (representing the cost of removing the "*p*" form "*comp* $\wedge$ *ng*" and having the "$\wedge$" substitute for "*mmuti*"). The string *VLDC* problem has been generalized for trees [35], introducing *VLDC* substitutions for whole paths or sub-trees.

Nonetheless, despite being comparable, it is clear that the repeatability and alternativeness operators in DTDs are fairly different from the *VLDC* symbols. *VLDC* symbols can replace any string (w.r.t. string matching) or sub-tree (w.r.t. tree matching) whereas the DTD operators specify constraints on the occurrence of a particular and well known node (and consequently the sub-tree rooted at that node). For example, in DTD of Figure 3.a, the operator "?" associated with element *length* (*length?*) designates that the specific node entitled *length* (and not any other node) can appear *0* or *1* time. Likewise for all DTD constraint operators.

2.2 Tree Edit Distance

In the past few years, tree edit distance algorithms, derived from edit distance between strings [20] [31] [32], have been extensively studied and exploited to compare semi-structured data, e.g. XML documents. Various methods for determining the structural similarities between Ordered Labeled Trees (OLTs) have been proposed. In essence, all these approaches aim at finding the cheapest sequence of edit operations that can transform one tree into another. Early approaches [29] [34] allow insertion, deletion and relabeling of nodes anywhere in the tree. However, they are relatively complex. For instance, the approach in [29] has a time complexity $O(|A||B|\ depth(A)\ depth(B))$ where $|A|$ and $|B|$ denote tree cardinalities while *depth(A)* and *depth(B)* are the depths of the trees. In [7] [9], the authors 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 by Chawathe [8] 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 [8]'s algorithm is of $O(N^2)$. Nierman and Jagadish [24] extend the approach in [8] by adding two new operations: insert tree and delete tree to allow insertion and deletion of whole sub-trees within in an OLT. Experimental results provided in

[24] show that their algorithm outperforms that of Chawathe [8], as well as [29]'s algorithm. The approach in [24] is of $O(N^2)$ complexity. A specialized version of [24] is provided in [10], where tree insertion/deletion costs are computed as the sum of the costs of inserting/deleting all individual nodes in the considered sub-trees (in comparison with [24], where certain sub-tree similarities are considered while assigning tree operations costs).

2.3 Similarity Between XML Documents and XML Grammars

Very few approaches have been developed to measure the structural similarity between XML documents and grammars. To our knowledge, the only methods are provided by Grahne and Thomo [15] and Bertino *et al.* [3] [4]. In [15], the authors address the problem of determining whether semi-structured data conform to a given data-guide, in the context of approximate querying. Note that this approach is not developed for XML documents and DTDs/schemas but for generic semi-structured data and data-guides (i.e. there are no constraints on the repeatability and alternativeness of elements). In addition, no experimental results assessing the performance of the approach were reported. Another approach provided by Bertino *et al.* [4] was dedicated to measuring the structural similarity between an XML document and a DTD. The proposed algorithm considers the level (i.e. depth) in which the elements occur in the hierarchical structure of the XML and DTD tree representations (in [3] [4], DTDs are modeled as labeled trees, similarly to XML documents) as well as element complexity (i.e. the cardinality of the sub-tree rooted at the element) when computing similarity values. The authors state that their approach is of exponential complexity but show that it becomes polynomial when the following assumption holds: *in the declaration of an element, two sub-elements with the same tag are forbidden*. In short, although the approach in [4] is proven effective in various situations [3], it is heuristic and time consuming.

3 Preliminaries

In this study, we focus on the XML/DTD comparison problem while aiming to achieve two specific targets: i) First of all, we intend to develop a fine-grained and precise method to compare XML documents and DTDs, w.r.t. the generic approach in [15] (disregarding DTD constraints) and heuristic method in [3], [4]; ii) Second, we aim to obtain an algorithm able to compare XML documents and DTDs in polynomial time, in comparison with the exponential method in [3], [4].

Making use of tree edit distance would help provide a structural comparison method that is more precise, more natural in the XML context, and that could be executed in polynomial time when comparing XML documents and DTDs, edit distance based methods being generally of quadratic ($O(N^2)$) or quadric ($O(N^4)$) complexity. In addition, a prominent advantage of using the edit distance as a similarity measure is that along with the distance value, a mapping between the nodes in the compared trees is provided in terms of the edit script (cf. Definition 4). The mapping can be visualized and can serve as an explanation of the similarity measure.

Several algorithms [5] have been developed to compute an edit distance, as the sum of a sequence of elemental edit operations (mainly node insertion, deletion and update) that can transform one tree structure into another (cf. Section 2). However, with DTD tree structures, constraints on the existence, repeatability and alternativeness of nodes come to play (mainly via the *?*, ***, *+*, *Or* operators). As a result, the problem of comparing an XML document to a DTD comes down to computing the edit distance between two trees, taking into account the various DTD constraint operators. In the following, we present some definitions and basic notions prior to developing our XML/DTD structural similarity approach.

3.1 Basic Definitions

Definition 1 - Ordered Labeled Tree: It is a rooted tree in which the nodes are ordered and labeled. We denote by *R(T)* the root node of tree *T*.

Definition 2 - Constraint operators: These are operators utilized in DTDs to specify constraints on the existence, repeatability and alternativeness of elements/attributes. With constraint operators, it is possible to specify whether an element is optional ('?'), whether it may occur several times ('*' for 0 or more times and '+' for 1 or more times), whether some sub-elements are alternative w.r.t. each other ("|" representing the *Or* operator) or are grouped in a sequence (',' representing the *And* operator). It is also possible to specify whether an attribute is optional ("*Implied*") or mandatory ("*Required*"). An element/attribute with no constraints (*null*) is mandatory and should appear exactly once.
Note that in this study, similarly to that of Bertino *et al.* in [3] [4], we do not associate constraint operators to expressions (e.g., *(a, b)**, *(a | c)+*, …) and focus on the case where operators are applied to elements only. This issue will be considered in upcoming studies.

Definition 3 - Lc-pair representation of a node: We define the lc-pair representation of a node as the pair *(l, c)* where: *l* is the node's label and *c* the associated constraint operator. We use *p.l* and *p.c* to refer to the label and the constraint operator of an *lc-pair* node *p* respectively.

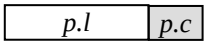


Fig. 1. Lc-pair graphical representation of an XML node *p*.

On the other hand, our edit distance XML structural similarity approach utilizes five edit operations, adopted from [8], [24]: *leaf node insertion*, *leaf node deletion* and n*ode update*, as well as *tree insertion* and *tree deletion*. Formal definitions are omitted due to lack of space.

Definition 4 - Edit script: It is a sequence of edit operations op_1, op_2, …, op_k. When applied to a tree *T*, the resulting tree *T'* is obtained by applying edit operations of the *Edit Script (ES)* to *T*, following their order of appearance in the script. By associating costs with each edit operation, $Cost_{Op}$, the cost of an *ES* is defined as the sum of the costs of its component operations: $Cost_{ES} = \sum_{i=1}^{|ES|} Cost_{Op_i}$.

Definition 5 - Edit distance: The edit distance between two trees *A* and *B* is defined as the minimum cost edit script that transforms *A* to *B*: $Dist(A, B) = Min\{Cost_{ES}\}$.
Consequently, the problem of comparing two trees *A* and *B*, i.e. evaluating the structural similarity between *A* and *B*, is defined as computing the corresponding tree edit distance [34].

3.2 Tree Structures

3.2.1 Tree Representation of XML Documents

An XML document is represented as a rooted ordered labeled tree (OLT) [33], where nodes stand for XML elements and are labeled with corresponding element tag names. Attributes appear as children of their encompassing elements, sorted by attribute name and appearing before all sub-element siblings. Similarly to [14], [24], element/attribute values are ignored since we are only interested in the structure of the document (cf. Figure 2).

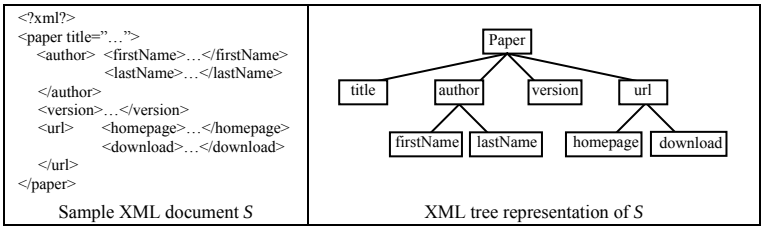


Fig. 2. Tree representation of an XML document.

3.2.2 Tree Representation of DTDs

As for XML documents, DTD can also be represented as a single tree or as a series of trees, depending on element constraints. In our approach, we represent a DTD lacking the '|' constraint operator (i.e. *Or*) as a single rooted ordered labeled tree (OLT), so as to attain a structure similar to that of XML document trees. The ',' operator (i.e. *And*) is mapped to its parent node in the document tree. Nodes grouped via the ',' operator are represented as the roots of the paths that are conjunctively connected. Constraint operators '?' and '*' are denoted via the *Lc-pair* representation of corresponding nodes (cf. Definition 3).

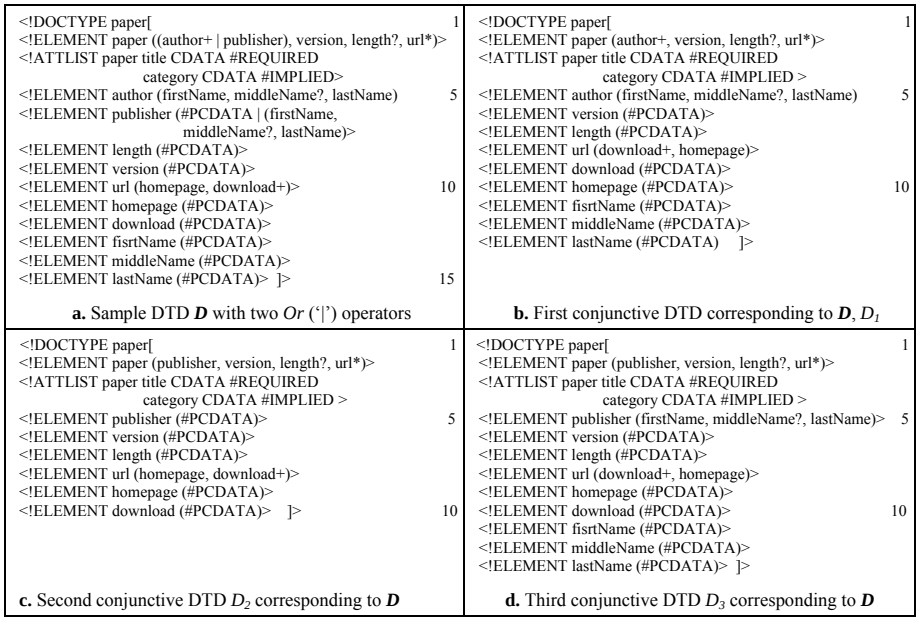


Fig. 3. A sample DTD including *Or* operators and its *original* and *disjunctive normal* forms.

An element assigned a "+" constraint operator (i.e. which would occur 1 or more times in the XML documents conforming to the DTD at hand) is represented as two consecutive nodes: one is mandatory (constraint = *null*, which would occur exactly once) and one is marked with a "*" constraint (which would occur 0 or many times, cf. Figure 4). Therefore, the "+" constraint operator itself does not appear in the tree representation of the DTD. Similarly, attribute constraints can be replaced by those applied on elements (*"Implied"* is replaced by *'?'* and *"Required"* by *null*, attributes being represented as sub-elements of their encompassing nodes). Element/attribute data types are ignored here as we are only interested in the structure of the DTD (cf. DTDs in Figure 3 and related tree representations in Figure 4).

When '|' comes to play, the DTD is decomposed into a *disjunctive normal form* [28]. The disjunctive normal form of a DTD is a set of conjunctive DTDs. Consider for example the DTD in Figure 3.a; it encompasses two *Or* operators. Thus, it can be represented as three separate conjunctive DTDs (cf. Figures 3.b, 3.c and 3.d) underlining the three possible structural configurations that could be attained via the different combinations of the *Or* operators. Consequently, the resulting conjunctive DTDs, that do not encompass *Or* ('|') operators, would be represented as OLTs, as discussed in the previous paragraph (cf. Figure 3).

Now that XML documents and DTDs are represented as OLTs, the problem of comparing an XML document and a DTD comes down to comparing the corresponding trees.

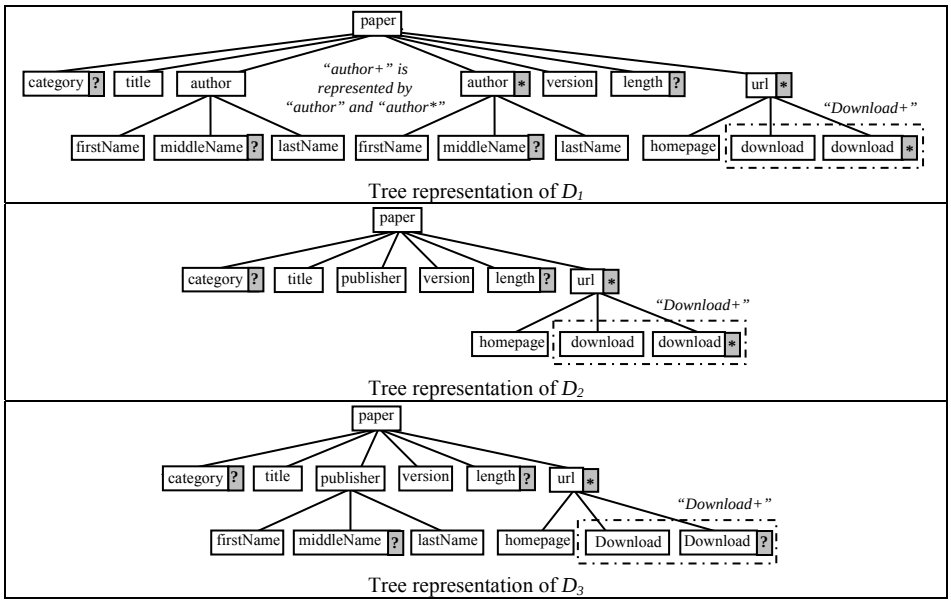


Fig. 4. Tree representations corresponding to DTD *D* of Figure 3.

4 Proposal

Our XML/DTD structural similarity approach consists of two phases: i) a pre-processing phase for transforming documents and DTDs into OLTs, ii) and an tree edit distance phase for computing the structural similarity between an XML document and a DTD. The overall algorithm, entitled *XDComparison*, is presented in Figure 5.

```
Algorithm XD_Comparison()
Input: DTD D and XML document S
Output: Similarity value between D and S ∈ [0,1]
Begin                                                                          1
    Begin Pre-processing phase
        DTreeSet = DTD_to_Tree(D)      // Generating the set of trees representing D
        STree = XML_to_Tree(S)         // Generating the tree representation of S
    End Pre-processing phase                                                   5
    Begin Edit distance phase
        Dist[] = new [1… |DTreeSet|]   // |DTreeSet| designates the number of trees
                                       // representing D (cf. Section 3.2)
        For (i=1 ; i ≤ |DTreeSet| ; i++)                                       10
        {
           Dist[i] = XD_EditDistance(DTreeSet[i] , STree)      // Edit distance algorithm
        }
        Return  1 / (1 + min(Dist[i]))     // Structural similarity value between D and S
    End Edit distance phase                                                    15
End
```

Fig. 5. The overall algorithm of our approach.

After transforming the DTD and the XML document into their tree representations (Figure 5, lines 1-5), the edit distance between each DTD tree (recall that multiple trees are obtained when the *Or* operator is utilized, otherwise, the DTD is represented by one single tree, cf.

Section 3.2) and the XML document tree is computed (Figure 5, lines 7-13). Subsequently, the minimum distance attainable is taken into account when computing the overall similarity value (Figure 5, line 14). Note that similarity measures based on edit distance are generally computed as *1/(1+Dist)* and vary in the *[0, 1]* interval. As a result, the maximum similarity (*=1*) in our case indicates that the concerned XML document is valid for the DTD to which it is compared.

While DTD/XML tree representations are central in our approach, the corresponding construction process is undertaken in average linear time. It is achieved by performing a pre-order traversal of the XML element/attributes in the file at hand, adding the corresponding tree nodes respectively. As for DTDs in particular, an additional tree instance is created every time an *Or* operator ('|') is encountered, the *And* operators (',') being mapped to their parent nodes. In addition, DTD tree nodes would encompass "?" and "*" constraint operators, the "+" operator being treated as a special case of "*" (detailed algorithms are disregarded here).

4.1 The Algorithm

Our edit distance algorithm for comparing a DTD tree and an XML document tree is given in Figure 6. It is based on a dynamic programming formulation, which is frequently used to solve minimum edit distance problems, and employs the edit operations mentioned in Section 3.1 (i.e. node insertion/deletion/update, as well as tree insertion/deletion). When the DTD tree is free of constraint operators (i.e. when all elements are mandatory), the algorithm comes down to a classical approximate tree matching process [24].

Note that when employing dynamic programming to compute the distance between a source tree (in our case the DTD tree) and a destination tree (in our case the XML document tree), a key issue is to first determine the cost of inserting every sub-tree of the destination tree and that of deleting every sub-tree of the source tree. An intuitive way of determining the cost of inserting (deleting) a sub-tree usually consists of computing the sum of the costs of inserting (deleting) all individual nodes in the considered sub-tree [10]. As for single node operations, unit costs are naturally utilized ($Cost_{Ins} = Cost_{Del} = 1$. $Cost_{Upd}(a, b) = 1$ when $a.l \neq b.l$, otherwise, $Cost_{Upd} = 0$, underlining that no changes are to be made to the label of node *a*). In this study, we restrict our presentation to the basic cost schemes above. Note that the investigation of alternative operations cost models, and how they affect our tree edit distance measure, is out of the scope of this paper. Such issues will be addressed in future studies.

Comparing an XML document and a DTD, in our approach, amounts to computing the edit distance between the corresponding trees, taking into account the "?", "*" and "+" DTD constraint operators. Recall that the *And*/*Or* operators are handled via our DTD tree model (cf. Section 3.2).

In order to consider, in our edit distance computations, that a DTD element with constraint operator "?" can only appear 0 or 1 time in the XML trees conforming to the DTD, a null cost is induced when the corresponding element node (as well as the sub-tree rooted at that node) is to be deleted (cf. Figure 6, lines 9-11 and 22-29). In other words, when an element assigned the "?" operator is to be deleted, the algorithm identifies the fact that its presence is optional and does not consider the corresponding deletion operation in the edit distance process.

In order to consider that a DTD element with the '*' constraint can appear 0 or many times in the XML trees conforming to the DTD, we proceed as follows:

- A null cost is induced when the corresponding element node (as well as the sub-tree rooted at that node) is to be deleted (cf. Figure 6, lines 9-11, 30-37), considering the fact that its presence is optional.
- When combining *insert tree*, *delete tree* and the recursive edit distance computations costs in order to determine the minimum cost script to be executed (cf. Figure 6, lines 30-37), the algorithm propagates the minimum distance value along with the recursive process (line 33). This allows to take into account the repeatability of the element that is assigned the '*' constraint without increasing the overall distance value.

As for the '+' operator, it is treated via the same processes dedicated to the '*' constraint operator. Since the '+' operator designates that the associated element can occur 1 or multiple times in the XML document, the designated element could be represented as two consecutive nodes: one that is mandatory (which will certainly appear in the document) and one marked with a '*' constraint (which would appear 0 or multiple times). Thus, as stated previously, the '+' constraint itself is disregarded in the tree representation of the DTD.

```
Algorithm XD_EditDistance()
Input: DTD tree D and XML tree S
Output: Edit distance between D and S
Begin                                                                          1
    M = Degree(D)                          // The number of first level sub-trees in A.
    N = Degree(S)                          // The number of first level sub-trees in B.
    Dist [][] = new [0...M][0…N]
    Dist[0][0] = Cost_Upd(R(D).l, R(S).l)   // R(D).l and R(S).l are the labels of the roots   5
                                            // of trees D and S
    For (i = 1 ; i ≤ M ; i++)
     {
        If (R(D_i).c = "?" or R(D_i).c = "*")   // R(D_i) is the root of 1st level sub-tree D_i in D
        {                                                                      10
            Dist[i][0] = Dist[i-1][0]    // no cost is added when "?" and "*" are encountered
        Else
            Dist[i][0] = Dist[i-1][0] + Cost_DelTree(D_i)   // Traditional tree edit distance.
        }
     }                                                                         15
     For (j = 1 ; j ≤ N ; j++) { Dist[0][j] = Dist[0][j-1] + Cost_InsTree(S_j) }
     For (i = 1 ; i ≤ M ; i++)
     {
        For (j = 1 ; j ≤ N ; j++)                                              20
        {
            If (R(D_i).c = "?")                 // The "?" operator is encountered
            {
                Dist[i][j] = min {
                    Dist[i-1][j-1] + XD_EditDistance(D_i, S_j),   // Dynamic programming.  25
                    Dist[i][j-1] + Cost_InsTree(S_j),             // Simplified tree
                    Dist[i-1][j]                                  //operation syntax.
                                }
            }
            Else If (R(D_i).c = "*")            // The "*" operator is encountered          30
            {
                Dist[i][j] = min{
                    min{ Dist[i-1][j-1], Dist[i][j-1] } + XD_EditDistance(D_i, S_j),
                    Dist[i][j-1] + Cost_InsTree(S_j),
                    Dist[i-1,j]                                                          35
                            }
            }
            Else                // no constraint operators (mandatory element/sub-tree)
            {
                Dist[i][j] = min{                                  // Traditional tree edit distance.  40
                    Dist[i-1][j-1] + XD_EditDistance(D_i, S_j),    // Dynamic programming.
                    Dist[i][j-1] + Cost_InsTree(S_j)               // Simplified tree
                    Dist[i-1][j] + Cost_DelTree(D_i),              // operations syntaxes.
                            }
            }                                                                    45
        }
    }
    Return Dist[M][N]
End
```

Fig. 6. Edit distance algorithm for comparing a DTD tree and an XML document tree.

4.2 Computation Example

We present the result of comparing sample XML document *S* in Figure 2 to the complex DTD *D* in Figure 3 in which the "|" (i.e. *Or*) operator is utilized. Following our approach, comparing

XML document *S* to DTD *D* comes down to comparing *S* to trees D_1, D_2 and D_3 which represent the *disjunctive normal form* of DTD *D* (*D* encompassing two *Or* operators). Distance values are reported below (detailed computations are omitted due to space limitations):

- $Dist(D_1, S) = 0$. *S* is valid for the DTD tree D_1
- $Dist(D_2, S) = 3$. Node *publisher* in D_2 is to be replaced by *author* in *S*. In addition, nodes *firstName* and *lastName* in *S* are not defined in D_2
- $Dist(D_3, S) = 1$. Node *publisher* in D_3 is to be replaced by *author* in *S*.

As stated previously, when a DTD corresponds to multiple conjunctive DTD trees, the minimum edit distance between those trees and the XML tree is retained. Consequently, the structural similarity between the DTD *D* and XML document *S* is computed as follows:

$$Sim(D,S) = \frac{1}{1 + min(Dist(D,S))} = \frac{1}{1 + min(Dist(D_1,S))} = 1$$ (maximum similarity value) which indicates that *S* conforms to *D*.

4.3 Overall Complexity

The computational complexity of our XML/DTD structural similarity approach simplifies to a polynomial formulation: $O(N^3)$. Let $|D|$ be the number of nodes (elements/attributes) in the DTD tree *D* considered (i.e. cardinality/size of *D*), $|D'|$ the cardinality of the largest conjunctive DTD corresponding to *D*, N_D the maximum number of conjunctive DTDs corresponding to *D*, and $|S|$ the cardinality of the XML tree *S* considered. The algorithm *XD_Comparison* (cf. Figure 5) is of complexity $O(|D|+|S| + (N_D |D'||S|)$:

- The pre-processing phase of *XD_Comparison* (cf. Figure 5, lines 2-5) is of average linear complexity and simplifies to $O(|D|+|S|)$.
- *XD_EditDistance*, which is of $O(|D'||S|)$ complexity, is executed for each conjunctive form of the DTD *D* (cf. Figure 5, lines 10-13). Thus, the complexity of the edit distance phase comes down to $O(N_D |D'||S|)$.

Note that the maximum number of *conjun*ctive DTDs, making up the *disjunctive normal form* of a generic DTD *D*, is $|D|-1$. This is obtained when the DTD has $(|D|-2)\times Or$ operators which is the maximum number of oper.tors that can occur in an XML definition. Therefore, in the worst case scenario, $O(|D|+|S|+N_D|D'||S|) = O(|D|+|S|+(|D|-1)|D'||S|)$ which simplifies to $O(N^3)$, where *N* is the maximum number of nodes in the DTD/XML trees being compared.

5 Experimental Evaluation

5.1 Evaluation Metrics

In order to validate our XML/DTD structural similarity approach, we make use of structural classification. In addition, we adapt to XML classification the *precision (PR)*, *recall (R)*, *noise* (*N*) and *silence* (*S*) measures, widely utilized in information retrieval evaluations, and propose a novel method for their usage in order to attain consistent experimental results (in the same spirit as Dalamagas *et al.*'s approach [10] w.r.t. XML document clustering). For an extracted class C_i that corresponds to a given DTD_i:

- a_i is the number of XML documents in C_i that correspond to DTD_i (correctly classified documents, i.e. documents that conform to DTD_i).
- b_i is the number of documents in C_i that do not correspond to DTD_i (mis-classified).
- c_i is the number of XML documents not in C_i, although they correspond to DTD_i (documents that conform to DTD_i and that should have been classified in C_i).

Therefore, $PR = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i}$, $R = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} c_i}$, $N = \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i}$ and $S = \frac{\sum_{i=1}^{n} c_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} c_i}$

where n is the total number of classes, which corresponds to the total number of DTDs considered for the classification task (PR, R, N, $S \in [0, 1]$ interval).

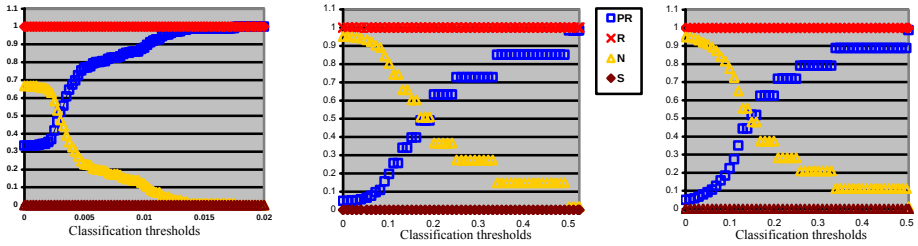
Thus, as with traditional information retrieval evaluation, high *precision*/*recall* and low *noise/silence* (indicating in our case high classification quality) characterize a good similarity method (structure-based XML/DTD similarity).

5.2 Classifying XML Documents

In our experiments, we undertook a series of multilevel classification tasks, varying the classification threshold in the *[0, 1]* interval:

- For the starting classification threshold ($s_1=0$), all XML documents appear in all classes, the starting classes.
- For the final classification threshold ($s_n=1$), with n the total number of classification levels (i.e. number of classification sets identified in the process), each class will only contain XML documents which actually conform to the DTD identifying the class.
- Intermediate classification sets will be identified for levels s_i / $s_1<s_i<s_n$.

Then, we compute *precision*, *recall*, *noise* and *silence* values for each of the classification sets identified in the multilevel classification phases, thus constructing *PR*, *R*, *N* and *S* graphs that describe the system's evolution throughout the classification process.



a. Classifying all 104 SIGMOD Record documents

b. Classifying XML documents of synthetic set 1 (*MaxRepeats*=5)

c. Classifying documents of synthetic set 2 (*MaxRepeats*=10)

Fig. 7. PR, R, N and S graphs for classifying real and synthetic XML documents

5.3 Results

We conducted experiments on real and synthetic XML documents. Two sets of 1000 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[3]. *Precision*, *recall, noise and silence* graphs are depicted in Figure 7.

One can clearly realize that *recall* (*R*) and *silence* (*S*) are always equal to *1* and *0* respectively. These values reflect the fact that our XML/DTD comparison algorithm constantly identifies in the DTD classes, regardless of the classification threshold, the XML documents that actually conform to the DTDs considered (i.e. documents and DTDs having *Sim(DTD, XML) =1*). On the other hand, *precision* (*PR*) gradually increases towards *1*, whereas *noise* (*N*) decreases to *0*, while varying the classification threshold from *0* to *1*:

- When the classification threshold is equal to 0, all documents in the XML repository are considered in each and every one of the classes corresponding to the DTDs at hand. That is underlined by minimum *precision* and maximum *noise* values.

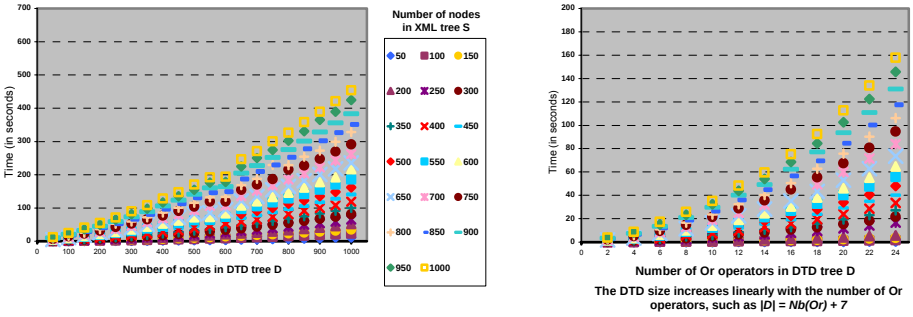
[1] http://www.xmlfiles.com and http://www.w3schools.com
[2] http://www.alphaworks.ibm.com
[3] http://www.acm.org/sigmod/xml

– Then, as the classification threshold increases, inconsistent documents are gradually filtered from the DTD classes, ultimately yielding classes that only encompass documents conforming to the considered DTDs.

PR and *N* results in Figure 7 show that our algorithm yields optimal classification quality, attaining optimal classes at a very early stage of the multilevel classification process (with classification thresholds < 0.5). Note that we do not experimentally compare our method to that of Bertino *et al* [4] since the authors do not provide the detailed algorithms of their approach.

5.4 Timing Analysis

We have shown, in Section 4.3, that the complexity of our XML/DTD structural comparison algorithm is polynomial following the size of the XML/DTD trees being compared, $O(|D|+|S|+|+(|D|-1)|D||S|)$ (which simplifies to $O(N^3)$, N being the maximum number of nodes in the XML/DTD trees S and D). This polynomial dependency on the size of each tree is experimentally verified, timing results being presented in Figure 8. The timing experiments were carried out on a PC with a Xeon 2.7 GHz processor (1GB RAM). Figure 8.a shows that the time to identify the structural similarity between XML trees and conjunctive DTD trees of various sizes grows in a linear fashion with tree size (underlining the complexity of *XD_EditDistance*, $O(|D'||S|)$). On the other hand, Figure 8.b shows that the time to execute our *XD_Comparison* algorithm grows in a polynomial fashion with DTD tree size when varying the number of *Or* operators (i.e. the number of conjunctive DTDs) in the DTD.



a. Timing results to compute pair-wise distances for XML/DTD trees (conjunctive DTDs) of various sizes.

b. Varying the number of *Or* operators (i.e. the number of conjunctive DTDs) in the DTD.

Fig. 8. Timing Results.

6 Conclusion

In this paper, we proposed a structure based similarity approach for comparing XML documents and DTDs. Based on the tree edit distance concept, our approach takes into account the various DTD constraints on the existence, repeatability and alternativeness of XML elements/attributes. Our theoretical study as well as our experimental evaluation showed that our approach is of polynomial complexity, in comparison with existing exponential algorithms, and that it yields optimal comparison results.

As continuing work, we are exploring the use of our approach in order to address, not only the structural aspect of XML documents (element/attribute labels), but also their information content (element/attribute values). By adding additional constraints on the data content of elements/attributes, DTDs could be exploited as *content-and-structure* queries, taking into account the structure of XML data in the search process, and returning ranked answers in the spirit of information retrieval (IR). Note that combining database (DB) structural "binary answer" XML search (e.g. XML-QL [11], XQuery [6]) and information retrieval query result ranking, is one of the most recent and promising trends in both DB and IR research. On the other hand, we plan to study XML/DTD comparison with DTDs including repeatable

expressions (expressions assigned '?', '*' and '+' constraint operators) as well as recursive definitions. In this context, it would be interesting to extend the tree model to a more general graph model for DTDs, and adapt (if possible) our tree edit distance algorithm accordingly.

References

[1] Aho A., Hirschberg D., and Ullman J., Bounds on the Complexity of the Longest Common Subsequence Problem. *ACM J.*, 23(1):1-12, January 1976.

[2] Akatsu T., Approximate String Matching with Don't Care Characters. *Information Processing Letters*, vol. 55, pp.235-239, 1995.

[3] Bertino E. *et al.*, Measuring the Structural Similarity among XML Documents and DTDs, Technical Report, University of Genova, 2002, http://www.disi.unige.it/person/MesitiM.

[4] 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*, 2004.

[5] Buttler D., A Short Survey of Document Structure Similarity Algorithms. In *Proc. of the 5th International Conference on Internet Computing*, pp. 3-9, 2004.

[6] Chamberlin D. *et al.*, 2001. XQuery: A Query Language for XML. http://www.w3.org/TR/2001/WD-xquery-20010215.

[7] Chawathe S., *et al.*, Change Detection in Hierarchically Structured Information, *SIGMOD*, pp. 493-504, 1996.

[8] Chawathe S., Comparing Hierarchical Data in External Memory. In *Proceedings of the 25th Int. Conf. on Very Large Data Bases*, pp. 90-101, 1999.

[9] Cobéna G. *et al.*, Detecting Changes in XML Documents. In *Proc. of the 18th ICDE Conference*, pp. 41-52, 2002.

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

[11] Deutsch A. *et al.*, XML-QL: A Query Language for XML. *Computer Networks*, 31(11-16):1155-1169, 1999.

[12] Do H.H. and Rahm E, COMA: A System for Flexible Combination of Schema Matching Approaches. In the *28th VLDB Conf.*, pp. 610-621, Honk Kong, 2002.

[13] Doan A., Domingos P. and Halevy A.Y., Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. *ACM SIGMOD,* pp. 509-520, 2001.

[14] Flesca S. *et al.*, Detecting Structural Similarities between XML Documents. *WebDB*, pp. 55-60, 2002.

[15] Goldman R. and Windom J., Dataguides: Enabling Query Formulation and Optimization in Semi-structured Databases. In *Proc. of the* 23rd *VLDB Conference*, pp. 436-445, 1997.

[16] Grahne G. and Thomo A., Approximate Reasoning in Semi-structured Databases. In *Proc. of the 8th International Workshop on Knowledge Representation meets Databases*, Vol. 45, 2001, http://CEUR-WS.org/Vol-45/03-thomo.ps

[17] Landau G. M. and Vishkin U., Fast Parallel and Serial Approximate String Matching. Journal of Algorithms, vol. 10, pp. 157-169, 1989.

[18] Lee J.H., Kim M.H. and Lee Y.J., Information Retrieval Based on Conceptual Distance in IS-A Hierarchies. *Journal of Documentation*, 49(2):188-207, 1993.

[19] Lee M. *et al.*, XClust: Clustering XML Schemas for Effective Integration. In *CIKM*, pp. 292-299, 2002.

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

[21] Madhavan J. *et al.*, Generic Schema Matching With Cupid. In *27th VLDB Conference*, pp.49-58, 2001.

[22] Melnik S., Garcia-Molina H. and Rahm E., Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proc. of the 18th ICDE Conference*, pp. 117-128, 2002.

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

[24] Nierman A. and Jagadish H. V., Evaluating structural similarity in XML documents. In *Proc. of the 5th Int. Workshop on the Web and Databases,* pp. 61-66, 2002.

[25] Pereira F., Technologies for Digital Multimedia Communications: An Evolution Analysis of MPEG Standards. *China Communications Journal*, pp. 8 – 19, 2006.

[26] Rijsbergen van C. J., *Information Retrieval*, Butterworths, London, 1979.

[27] Sanz I., Mesiti M., Guerrini G. and Berlanga Lavori R., Approximate Subtree Identification in Heterogeneous XML Documents Collections. *XSym*, pp. 192-206, 2005.

[28] Schlieder T., Similarity Search in XML Data Using Cost-based Query Transformations. In *WebDB*, pp. 19-24, 2001.

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

[30] Tai K.C., The Tree-to-Tree correction problem. *ACM J.*, 26, pp.422-433, 1979.

[31] Wagner J. and Fisher M., The String-to-String correction problem. *J. of the ACM*, (21)168-173, 1974.

[32] Wong C. and Chandra A., Bounds for the String Editing Problem. Journal of the Association for Computing Machinery, 23(1):13-16, January 1976.

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

[34] Zhang K. and Shasha D., Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM J.*, 18(6):1245-1262, 1989.

[35] Zhang K., Shasha D. and Wang J., Approximate Tree Matching in the Presence of Variable Length Don't Cares. *J. of Algorithms*, 16(1):33-66, 1994.

[36] Zhang Z., Li R., Cao S. and Zhu Y., Similarity Metric for XML Documents. *Knowledge Management and Experience Management Workshop*, Karlsruhe, Germany, 2003.