Matching an XML Document against a Set of DTDs

Elisa Bertino[1] *Giovanna Guerrini*[2] *Marco Mesiti*[2]

[1] Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano, Italy
`bertino@dsi.unimi.it`
[2] Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova, Italy
{`guerrini,mesiti`}`@disi.unige.it`

Abstract. *Sources of XML documents are proliferating on the Web and documents are more and more frequently exchanged among sources. At the same time, there is an increasing need of exploiting database tools to manage this kind of data. An important novelty of XML is that information on document structures is available on the Web together with the document contents. However, in such an heterogeneous environment as the Web, it is not reasonable to assume that XML documents that enter a source always conform to a predefined DTD in the source. In this paper we address the problem of document classification by proposing a metric for quantifying the structural similarity between an XML document and a DTD. Based on such notion, we propose an approach to match a document entering a source against the set of DTDs available in the source, determining whether a DTD exists similar enough to the document.*

1 Introduction

XML [10] has recently emerged as the most relevant standardization effort for document representation and exchange on the Web. It offers the possibility of defining tags and modeling nested document structures. XML documents, thanks to semantic tags, are self-describing. Thus, with the advent of XML, information on document structures (provided through the tags embedded in a document) are available on the Web. These information can obviously be used in order to improve document retrieval. XML also offers the possibility of defining document types (called DTDs -*Document Type Definitions*) that describe the structure of documents. An XML DTD essentially is a grammar constraining the tags and the structure of a document. A document whose structure conforms to a DTD is called *valid* in XML terminology. Obviously, the presence of a DTD allows one to take advantage of the knowledge about document structures for storing, querying, and indexing a set of documents.

In such an heterogeneous and flexible environment as the Web, however, we cannot assume that documents related to the same topic, that is, "talking of the same things", exactly have the same structure. Thus, we cannot fix a certain set of predefined DTDs and then only handle XML documents that are valid for a DTD in that set. Similarly, if we aim at developing an XML-based search engine capable of extracting from the Web those (portions of) documents dealing with given semantic structural properties, and we express the user query as a DTD, we cannot retrieve only the documents valid for that DTD.

We thus address the problem of matching documents against a set of DTDs even when such documents do not fully conform to any DTD in such set. Specifically, we allow some attributes and subelements specified for an element in the DTD to be missing in the corresponding element of the document, and, viceversa, we allow the document to contain some additional attributes and subelements not appearing in the DTD. Moreover, we allow elements/attributes in the document to follow a different order w.r.t. the one specified in the DTD (i.e. we are focusing on data-centric documents). Finally, document and DTD tags could not be exactly the same, provided they are *stems* or are similar enough according to a given Thesaurus. In matching a document against a DTD the goal is then to quantify, through an appropriate measure, the structural similarity between the document and the DTD.

The scenario we refer to consists of a number of heterogeneous sources of XML documents able to exchange documents among each other. Each source stores and indexes the local documents according to a set of local DTDs. An XML document entering a source is matched against the DTDs in the source. If a DTD exists in the source to which the document conforms according to the usual notion, then the document is accepted as valid for this DTD. Otherwise, the proposed metric is used for selecting the DTD, among the ones in the source, that best describes the document structure.

The matching process is performed against a tree representation of both documents and DTDs. This representation can be easily obtained from their declarations. Though our technique handles all the features of XML documents, in the paper we will focus on the most meaningful *core* of the approach, thus we restrict ourselves to a subset of XML documents and to tag equality. We will only briefly sketch how the approach generalizes to arbitrary XML documents and how tag similarity is handled, and we refer the interested reader to [2] for the treatment of the general case.

Our proposal is, to the best of our knowledge, the first one addressing the issue of measuring the structural similarity of XML documents and DTDs, and it has relevant potential applications for structural clustering of documents and for document retrieval based on conditions on the document structure. Our work benefits from previous work developed in the context of automatic object classification [1] and of heterogeneous information integration (both at the data [6] and at the schema [3] levels).

The remainder of the paper is structured as follows. Section 2 presents our tree representation of XML documents and DTDs. Section 3 presents the similarity measure and the matching process. Finally, Section 4 concludes the work, by discussing extensions and applicability of the proposed technique.

```
<product>
  <name>Deliver</name>
  <urls>
   <download>
     http://.../deliver.tgz
   </download>
   <homepage>
     http://.../index.html
   </homepage>
  </urls>
  <description>Mail... </description>
  <author>
   <fName>Chip</fName>
   <lName>Salzenberg</lName>
  </author>
  <version>2.1.13</version>
</product>
```

(a)

```
<!DOCTYPE product[
<!ELEMENT product(name,urls+,description,
          (author*|vendor),version?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT urls(download*, homepage)>
<!ELEMENT description ANY>
<!ELEMENT author (fName, mName?, lName)>
<!ELEMENT vendor (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT download (#PCDATA)>
<!ELEMENT homepage (#PCDATA)>
<!ELEMENT fName (#PCDATA)>
<!ELEMENT mName (#PCDATA)>
<!ELEMENT lName (#PCDATA)>
]>
```

(b)

Fig. 1: An example of document and DTD

2 Documents and DTDs as Trees

Fig. 1(a) shows an example of XML document describing software products whereas Fig. 1(b) shows a possible corresponding DTD. The example shows that, in a DTD, for each subelement it is possible to specify whether it is optional (‘?’), whether it may occur several times (‘*’ or ‘+’ with the usual meaning), and whether some subelements are alternative with respect to each other (‘|’). As we have anticipated in the introduction, we will focus on a subset of XML documents. In particular, we will only consider elements (that can have a nested structure) disregarding attributes (that can be seen as a particular case of elements). Since we disregard attributes, we will only consider nonempty elements. Moreover, since we focus on data-centric documents, we disregard the order of subelements. In the matching process, we represent both DTDs and XML documents through labeled trees. The document representation is compliant wit the tree representation of DOM [9]. By contrast, the representation of DTDs we adopt facilitates the description of the algorithms we propose. In this section we discuss this representation.

2.1 Tree Representation of Documents

An XML document is represented as a labeled tree. Our representation is based on the classical definition of labeled tree.[1] In our representation of documents each node represents an element or a value. The label associated with a node represents the corresponding tag name or value. The labels used to label the tree belong to a set of element tags ($\mathcal{EN}$) and to a set of values that the data content of an element can assume ($\mathcal{V}$). In each tree that represents an XML document the root label belongs to $\mathcal{EN}$ (it is the document element name). Moreover, all nodes

[1] Given a set $\mathcal{N}$ of nodes, a tree is defined by induction as follows: $v \in \mathcal{N}$ is a tree; if $T_1, \ldots, T_n$ are trees, then $(v, [T_1, \ldots, T_n])$ is a tree. A label is then associated with each node of the tree from a set $\mathcal{A}$ of labels.

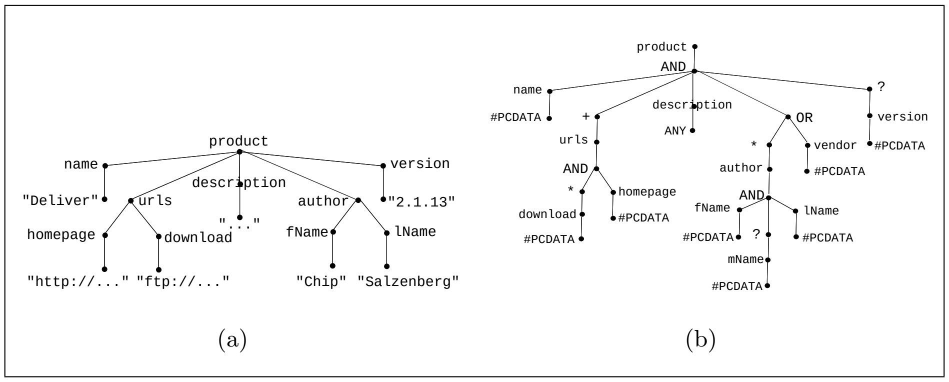

Fig. 2: Tree representations of a document and a DTD

labeled by values in $\mathcal{V}$ are tree leaves. Fig. 2(a) shows the tree representation of the XML document of Fig. 1(a).[2]

2.2 Tree Representation of DTDs

A DTD is also represented as a labeled tree. In the tree representation, in order to represent optional elements, repeatable elements, and alternative of elements, the set of operators $\mathcal{OP} = \{$`AND`, `OR`, ?, $*\}$ is introduced. The `AND` operator represents a sequence of elements, the `OR` operator represents an alternative of elements, the `?` operator represents an optional element, and the `*` operator represents repeatable elements (0 or more times). We remark that there is no need to introduce the `+` operator because an element type declaration like <`!ELEMENT a (b)+`> can be replaced by the equivalent one <`!ELEMENT a (b,b*)`>.

In our representation of DTDs each node corresponds to an element tag, to an element type, or to an operator. In each tree that represents a DTD there is a single edge outgoing from the root, and the root label belongs to $\mathcal{EN}$ (it is the name of the main element of documents described by the DTD). Moreover, there can be more than one edge outgoing from a node, only if the node is labeled by `AND` or `OR`. Finally, all nodes labeled by types are leaves of the tree. Fig. 2(b) shows the tree representation of the `product` DTD of Fig. 1(b).

We remark that the introduction of operators $\mathcal{OP} = \{$`AND`, `OR`, ?, $*$, $+\}$ allows us to represent the structure of all kinds of DTDs. The introduction of the `AND` operator is required in order to distinguish between an element containing an alternative between sequences (e.g. <`!ELEMENT a(b|(c1,c2))`>) and an element containing the alternative between all the elements in the sequence (e.g. <`!ELEMENT a(b|c1|c2)`>).

Note finally that not only a document/DTD is a tree, but any element of a document/DTD is a tree, subtree of the document/DTD. For instance, element `name` of the document in Fig. 1(a) corresponds to the leftmost subtree, whose root is reached by an edge from the `product` node in Fig. 2(a).

[2] Explicit direction of edges is omitted. All edges are oriented downward.

3 Measuring Similarity

In this section we first outline the requirements that a similarity measure should meet in order to properly evaluate the similarity between an XML document and a DTD, and then present a similarity measure addressing such requirements. Specifically, we describe the evaluation function we employ, then present the matching algorithm, and finally define the similarity measure.

3.1 Requirements for a Similarity Measure

We briefly introduce the requirements for a metric to measure the structural similarity between a document and a DTD.

Common, Plus, and Minus Elements. The similarity measure should consider: elements appearing both in the document and in the DTD, referred to as *common* elements; elements appearing in the document but not in the DTD, referred to as *plus* elements; elements appearing in the DTD but not in the document, referred to as *minus* elements. Obviously, to achieve the best similarity, plus and minus elements should be minimized and common elements should be maximized.

Repeatable Elements. There can be many ways in which a document can be matched against a DTD. Because of alternative and repeatable elements, indeed, a DTD describes different possible document structures. When matching a document against a DTD, a structure, among those possible structures, must be identified, that is the most adequate for the document. In case of repeatable elements, the similarity measure must identify the best number of repetitions, that is, the one that maximizes common elements and minimizes plus and minus elements. Note that an higher number of repetitions can allow every element in the document to match with an element in the DTD (no plus) but can increase the number of unmatched elements in the DTD (minus).

Level of an Element. Elements in a document are not all “equivalent”: elements at higher levels in the document structure are more relevant than subelements deeply nested in the document structure. The similarity measure should catch the intuition that the elements at a higher level in a document are more relevant than the elements at a lower level.

We thus introduce the notion of level of an element, related to the depth of the corresponding tree. Given a tree T in a tree representing a document, the level of T is its depth as a tree, that is, the number of nodes along the longest maximal path (that is, a path from the root to a leaf) in T. By contrast, given a tree T in a tree representing a DTD, its level is the number of nodes, not labeled by an operator, along the longest maximal path in T. Nodes labeled by operators in DTD trees, indeed, only influences the breadth of the corresponding document trees, not their depths. In the following, given a tree T, $label(T)$ denotes the label of the root of T.

Definition 1. (Function *Level*). *Let $T = (v, [T_1, \ldots, T_n])$ be a subtree of a document or a DTD. Function Level is defined as follows:*

$$Level(T) = \begin{cases} 1 + max_{i=1}^{n} Level(T_i) & \text{if } label(T) \in \mathcal{EN} \\ max_{i=1}^{n} Level(T_i) & \text{if } label(T) \in \mathcal{OP} \\ 0 & \text{otherwise} \end{cases}$$ □

Example 1. Let T denote the tree of Fig. 2(b), then $Level(T) = 3$. ○

Now we can assign a different weight to elements at different levels of the tree. Let $l = Level(T)$ be the level of a document/DTD tree T, the root of T will have weight γ^l, and the weight is then divided by γ when going down a level. Thus, γ^{l-i} is the weight of a generic level i of T. $\gamma \in \mathbb{N}$ is a parameter of our technique, that allows one to specify the relevance of information at higher levels in the document with respect to information at lower levels. By taking $\gamma = 1$ all the information are considered equally relevant, and thus the fact that elements appear at different levels in the nested structure is not taken into account. In what follows, we will consider $\gamma = 2$. In this way, the relevance of each node is double than the relevance of its children.

Weight of an Element. The similarity measure should also take into account the fact that an element with a more complex structure can appear in the document when a simple data element is declared in the DTD, or that, by contrast, the element in a document can have a structure simpler than the one specified for the corresponding element in the DTD.

The similarity measure should consider the structure of plus and minus elements. In case of minus elements, however, the structure is not fixed. Our idea is to consider, as structure of the minus elements, the simplest structure associated with that portion of DTD. Thus, the measure should not take into account optional or repeatable elements and, in case of alternative, the measure should take into account only one of the alternative elements (the one with the simplest structure).

We thus introduce function $\mathcal{W}eight$ to evaluate a subtree of a document or of a DTD. This function is used to measure the "unmatched" components (plus and minus) in the matching process. Given a subtree of the document and a weight w, function $\mathcal{W}eight$ multiplies the number of elements in each level for the weight associated with the level. The resulting values are then summed. Given a subtree of the DTD and a weight w, function $\mathcal{W}eight$ works as on a document, but it takes into account only mandatory elements in the DTD. That is, the function does not consider optional elements or repeatable elements. Moreover, in case of `OR` labeled nodes, the weights associated with the possible alternatives are evaluated and the minimal value is chosen. The choice of the minimal value corresponds to select the subtree with the simplest structure.

Definition 2. (Function $\mathcal{W}eight$). *Let T be a subtree of a document or a DTD, and w_l be the weight associated with the level of T in D. Function $\mathcal{W}eight$ is* *defined as follows:*

$$\mathcal{W}eight(T, w_l) = \begin{cases} w_l & \textit{if label}(T) \in \mathcal{V} \cup \mathcal{ET} \\ 0 & \textit{if label}(T) \in \{*, ?\} \\ \sum_{i=1}^{n} \mathcal{W}eight(T_i, w_l) & \textit{if label}(T) = \texttt{AND},\ T = (v, [T_1, \ldots, T_n]) \\ min_{i=1}^{n} \mathcal{W}eight(T_i, w_l) & \textit{if label}(T) = \texttt{OR},\ T = (v, [T_1, \ldots, T_n]) \\ \sum_{i=1}^{n} \mathcal{W}eight(T_i, \frac{w_l}{\gamma}) + w_l & \textit{otherwise, where } T = (v, [T_1, \ldots, T_n]) \end{cases}$$

□

Example 2. Let T denote the tree of Fig. 2(b), recall that we consider $\gamma = 2$, $\mathcal{W}eight(T, 8) = 20$. Note that, in this example, the level weight 8 is 2^3, where 3 is the number of levels of the DTD. ○

3.2 Evaluation Function

Having outlined the requirements of the similarity measure, we introduce an evaluation function that allows one to quantify the similarity degree between a document and a DTD. The function takes as input a triple of integer values (p, m, c). The (p, m, c) triple represents the evaluation of plus, minus, and common elements, taking also into account their different levels in the document and in the DTD. These three values are combined to obtain a similarity evaluation.

Definition 3. *(*Function $\mathcal{E}$*). Let (p, m, c) be a triple of natural numbers and α, β be real numbers s.t. $\alpha, \beta \geq 0$. Function $\mathcal{E}$ is defined as follows:*

$$\mathcal{E}((p, m, c)) = \frac{c}{\alpha * p + c + \beta * m}$$

□

Function $\mathcal{E}$, defined according to the *ratio model* [8], is based on two parameters α, β and returns a value between 0 and 1. Depending on the value assigned to these parameters, the function gives more relevance to *plus* elements w.r.t. *minus* elements, or vice-versa. For example, if $\alpha = 0$ and $\beta = 1$ the function does not take into account *plus* elements in measuring similarity. Therefore, a document with only extra elements w.r.t. the ones specified in the DTD has a similarity measure equal to 1. By contrast, if $\alpha = 1$ and $\beta = 0$ the evaluation function does not take into account *minus* elements in the similarity measure. In the following examples we assume that $\alpha = \beta = 1$, thus giving the same relevance to *plus* and *minus* elements.

3.3 Matching Algorithm

An algorithm, named $\mathcal{M}atch$, that allows one to assign a (p, m, c) triple to a pair of trees $(document, DTD)$ has been defined [2]. Such algorithm is based on the idea of locally determining the best structure for a DTD element, for elements containing alternatives or repetitions, as soon as the information on the structure of its subelements in the document are known.

The algorithm is general enough to evaluate the similarity between any kind of XML documents and DTDs. In this paper, however, we focus on the most meaningful *core* of the algorithm, based on the assumption that, in the declaration of an element, two subelements with the same tag are forbidden. That is, element declarations such as `<!ELEMENT a (b*, (c|b))>` are not considered. The general version of the algorithm is briefly discussed in Section 4 and all the details can be found in [2].

Given a document D, and a DTD T, algorithm $\mathcal{M}atch$ first checks whether the root labels of the two trees are equal. If not, then the two structures do not have common parts, and a null triple is returned. If the root labels are equal, the maximal level l between the levels of the two structures is determined, and the recursive function $\mathcal{M}$ is called on: (1) the root of the document, (2) the first (and only) child of the DTD, (3) the level weight (2^{l-1}) considering the fact that function $\mathcal{M}$ is called on the second level of the DTD structure, (3) a flag indicating that the current element (the root element) is not repeatable.

Function $\mathcal{M}$ recursively visits the document and the DTD, at the same time, from the root to the leaves, to match common elements. Specifically, two distinct phases can be distinguished:

1. in the first phase, moving down in the trees from the roots, the parts of the trees to visit through recursive calls are determined, but no evaluation is performed;
2. when a terminal case is reached, on return from the recursive calls and going up in the trees, the various alternatives are evaluated and the best one is selected.

Intuitively, in the first phase the DTD is used as a “guide” to detect the elements of the document that are covered by the DTD, disregarding the operators that bind together subelements of an element. In the second phase, by contrast, the operators used in the DTD are considered in order to verify that elements are bound as prescribed by the DTD, and to define an evaluation of the missing or exceeding parts of the document w.r.t. the DTD. Terminal cases are the following: a leaf of the DTD is reached, or an element of the DTD not present in the document is found. In these cases a (p, m, c) triple is returned. Therefore, the second phase starts and the evaluation of internal nodes is performed, driven by their labels.

For lack of space, we do not include the definition of function $\mathcal{M}$, rather we illustrate its behavior on the document and the DTD in Fig. 3(a,b). For sake of clarity, in the discussion of the algorithm, we denote the element of the document labeled by $\mathtt{a}$ as $\mathtt{a}_D$, and the element of the DTD labeled by $\mathtt{a}$ as $\mathtt{a}_T$.

During the first phase, function $\mathcal{M}$, driven by the label of the current DTD node, is called on subtrees of the document and the DTD. For example, on the first call of $\mathcal{M}$ on ($\mathtt{a}_D$, $\mathtt{AND}_T$), recursive calls on $\mathtt{a}_D$ and all the subtrees of $\mathtt{AND}_T$ are performed (i.e., on ($\mathtt{a}_D$, $\mathtt{b}_T$), and ($\mathtt{a}_D$, $\mathtt{AND}_T$)). Recursive calls are performed disregarding the operators in the DTD and moving down only when an element

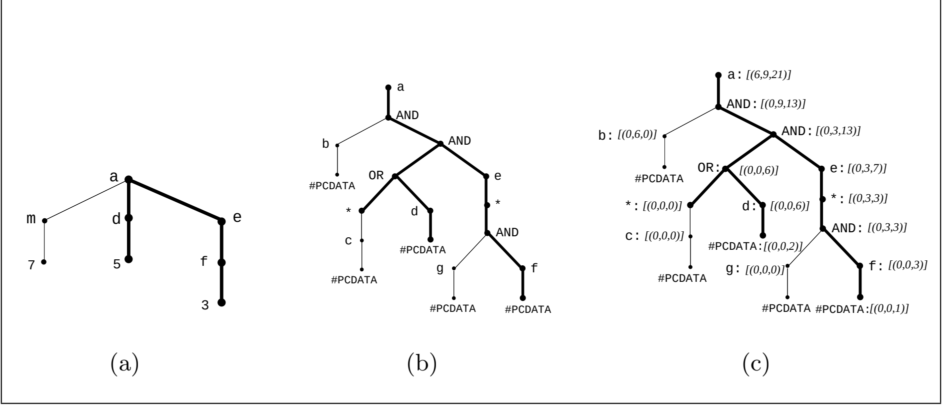

Fig. 3: Execution of function $\mathcal{M}$

declared in the DTD is found in the document as child of the current node. Moreover, in such cases, the weight level is divided by γ in order to determine the level weight of the underlying level. Fig. 3(a,b) shows the recursive calls performed. An edge (v, v') of the tree is bold if a recursive call of function $\mathcal{M}$ has been made on the subtree rooted at v'. Note that no recursive calls have been made on $\mathtt{b}_T$, $\mathtt{c}_T$, and $\mathtt{g}_T$ because such elements are not present in the document. Note also that $\mathtt{m}_D$ has not been visited by function $\mathcal{M}$, because this element is not required in the DTD.

When a terminal case is reached, a (p, m, c) triple is produced. For example, when function $\mathcal{M}$ is called on ($\mathtt{f}_D$, $\mathtt{\#PCDATA}_T$), the triple $(0, 0, 1)$ is generated, because the DTD requires a data content for $\mathtt{f}_D$ and, actually, such element has a textual content. By contrast, when function $\mathcal{M}$ is called on ($\mathtt{a}_D$, $\mathtt{b}_T$), the triple $(0, 6, 0)$ is generated, because the DTD requires an element tagged `b`, but such element is missing in the document. Therefore, function $\mathcal{W}eight$ is called on $\mathtt{b}_T$ and, since the current level weight is 4, the value 6 is returned as weight of the missing subtree.

On return from the recursive calls, the operators and the repeatability of the node are considered in order to select the best choice among the possible ones for binding together subelements. For example, returning from the evaluation of subtrees of the `OR` element, which is not repeatable, the triples $(0, 0, 0)$ and $(0, 0, 6)$ obtained for the evaluation of subtrees are considered. The best one is selected relying on the $\mathcal{E}$ evaluation function. By contrast, returning from the evaluation of subtrees of an `AND` element, which is not repeatable, the obtained evaluations are summed in order to determine the evaluation of the sequence of elements. The behavior of the algorithm is much more articulated when elements are repeatable. In such cases, indeed, not only a triple is generated, but a list of triples. The lists of triples are combined in order to evaluate internal nodes.

The intermediate evaluations are reported in Fig. 3(c). If an edge is bold the label is followed by the (p, m, c) triple obtained from the evaluation of the corresponding subtree. If an edge is not bold, but the label is followed by a (p, m, c) triple, it represents the evaluation of minus elements of the subtree.

3.4 Similarity Measure

The similarity measure between a document and a DTD is defined as follows.

Definition 4. *(*Similarity Measure*). Let D be a document and T a DTD. The similarity measure between D and T is defined as follows:*

$$\mathcal{S}(D, T) = \mathcal{E}(\mathcal{M}atch\langle D, T\rangle) \qquad \square$$

Example 3. Let D and T be the document and the DTD of Fig. 3(a,b). Their similarity degree is $\mathcal{S}(D, T) = \mathcal{E}(\mathcal{M}atch\langle D, T\rangle) = \mathcal{E}(\langle 6, 9, 21\rangle)$=0.58. $\bigcirc$

The following proposition states the relationship between the notion of validity and our similarity measure.

Proposition 1. *Let D be a document, T a DTD, and α, β the parameters of function $\mathcal{E}$. If $\alpha, \beta \neq 0$ the following properties hold:*

- *If D is valid w.r.t. T, then $\mathcal{S}(D, T) = 1$;*
- *If $\mathcal{S}(D, T) = 1$, then D is valid w.r.t. T, disregarding the order of elements.* $\triangle$

4 Discussion

Though we have presented a *core* of the algorithm that works for a subset of XML documents, the approach can handle arbitrary XML documents [2]. Attribute handling does not raise new relevant issues. Subelements with the same tags are handled by considering and evaluating all the possible matching between different subelements with the same tag in the DTD and an element with that tag in the document. A matrix is built for each element in the DTD with some direct subelements with the same tag. This matrix has a column for each of the subelements of an element with the same tag, and it contains a row for each of the possible ways in which the elements with that tag at that level in the document can be matched with them. The similarity measure is evaluated for all these possible matching, and the best one is then selected.

Tag similarity can be evaluated relying on a Thesaurus [5], containing a set of terminological relationship, among which, for instance, SYN and USE. Different affinity values $\delta \in [0, 1]$ can be assigned to each terminological relationship. In the similarity measure, when matching two element tags, the value $1 - \delta$ is added to the component m of the subtree evaluation. In this way we capture the missing tag equality.

The complexity of the algorithm we have presented in Section 3 is polynomial in the number of nodes of the document and of the DTD. Each node in the document and each node in the DTD is indeed visited only once, and for each node in the DTD, in the worst case, some operations that are quadratic in the number of nodes of the document and DTD are performed. In the general case discussed above, however, the algorithm is exponential. Thus, there is an obvious trade-off between the efficiency of the matching process and the strictness of the similarity requirements. Details on the algorithm complexity are reported in [2].

The described algorithm as well as its extensions discussed above have been implemented in Java, using the DOM libraries [9]. To validate the proposed technique we have performed some experiments over "real data", gathered from the Web, and "synthetic data", randomly generated. Because of space limitations, we cannot report on these experiments here and we refer to [2]. In both the experiments, however, we obtained that for each document D, and for each pair of DTDs T_1, T_2 such that D is not valid neither for T_1 nor for T_2, whenever $\mathcal{S}(D, T_1) > \mathcal{S}(D, T_2)$, D actually is more similar to T_1 than to T_2.

The work described in this paper can be extended in different directions. First, we are investigating how to use the feedback information of the matching algorithm to obtain a set of DTDs describing in the most accurate way the structure of the considered XML documents. Another direction we are investigating is the integration of structure-based retrieval with classical content-based retrieval. The goal is to express and evaluate queries containing both filters on the document structure and on the document content. In this context, we also plan to investigate how the defined structural similarity measure can be employed for structural clustering of XML documents.

References

1. E. Bertino, G. Guerrini, I. Merlo, and M. Mesiti. An Approach to Classify Semi-Structured Objects. In *Proc. European Conf. on Object-Oriented Programming*, LNCS 1628, pp. 416–440, 1999.
2. E. Bertino, G. Guerrini, and M. Mesiti. Measuring the Structural Similarity among XML Documents and DTDs, 2001. http://www.disi.unige.it/person/MesitiM.
3. S. Castano, V. De Antonellis, M. G. Fugini, and B. Pernici. Conceptual Schema Analysis: Techniques and Applications. *ACM Transactions on Database Systems*, 23(3):286–333, Sept. 1998.
4. M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. of Int'l Conf. on Management of Data*, pp. 165–176, 2000.
5. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, Nov. 1995.
6. T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proc. of Int'l Conf. on Very Large Data Bases*, pp. 122–133, 1998.
7. S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In *Proc. of Int'l Conf. on Management of Data*, pp. 295–306, 1998.
8. A. Tversky. Features of Similarity. *J. of Psychological Review*, 84(4):327–352, 1977.
9. W3C. Document Object Model (DOM), 1998.
10. W3C. Extensible Markup Language 1.0, 1998.