J Intell Inf Syst (2008) 30:55–92
DOI 10.1007/s10844-006-0023-y

Measuring the structural similarity among XML documents and DTDs

Elisa Bertino · Giovanna Guerrini · Marco Mesiti

Received: 24 September 2004 / Revised: 16 February 2006 /
Accepted: 22 February 2006 / Published online: 27 January 2007

Abstract Measuring the structural similarity between an XML document and a DTD has many relevant applications that range from document classification and approximate structural queries on XML documents to selective dissemination of XML documents and document protection. The problem is harder than measuring structural similarity among documents, because a DTD can be considered as a generator of documents. Thus, the problem is to evaluate the similarity between a document and a set of documents. An effective structural similarity measure should face different requirements that range from considering the presence and absence of required elements, as well as the structure and level of the missing and extra elements to vocabulary discrepancies due to the use of synonymous or syntactically similar tags. In the paper, starting from these requirements, we provide a definition of the measure and present an algorithm for matching a document against a DTD to obtain their structural similarity. Finally, experimental results to assess the effectiveness of the approach are presented.

Keywords H.3.2 Information storage · H.3.3 Information search and retrieval · H.3.5.f XML/XSL/RDF · I.5.3.b Similarity measure

E. Bertino
Purdue University, USA

G. Guerrini
University of Genova, Italy

M. Mesiti (✉)
University of Milano, Italy
e-mail: mesiti@dico.unimi.it

1 Introduction

XML (*eXtensible Markup Language*) (W3C, 1998) is a markup language that has emerged as the most relevant standardization effort for document representation and exchange on the Web. Since XML provides data description features that are very similar to those of advanced data models, XML is today supported either as native data model or on top of a proprietary data model by several database management systems. As a result, XML databases on the Web are proliferating. The availability of structural and schema information is crucial in providing efficient and effective XML data handling with classical database functionalities.

A major goal is now to make such databases able to easily interoperate in order to enable information and knowledge sharing across the Web. The use of a common data representation model, like XML, greatly simplifies the task of interoperation and integration among heterogeneous data management systems. However, we cannot reasonably expect XML to solve all the issues related to semantic heterogeneity of data. The highly dynamic nature of the Web and of the information sources available on it, together with the large number of these sources, make it impossible for a site to fully agree in advance on the schema with all possible data sites with which exchanges may take place, despite the efforts towards the development of standardized schemas (XML.org, 2003). Thus, a source may contain a schema describing the structure of a certain kind of documents, and it may happen to handle documents of this kind that, however, do not exactly conform to the schema. There is therefore a strong need for methodologies and tools able to model and evaluate the structural similarity between an XML document and a DTD in order to check whether the structure of a document is similar enough to a DTD to be considered a weak instance of it, and to identify their common and divergent components.

In our work we focus on data centric documents (Bourret, 1999), that is, documents presenting a possibly highly nested structure and representing specific *data* (e.g., department employees, financial records, movie information, etc.). Thus, the DTD order constraint is not relevant in our context. Note that, keeping order into account reduces the complexity of evaluating the similarity between the two structures, thus, disregarding order, we are addressing a more difficult problem.

As discussed in Bertino, Guerrini, and Mesiti (2004a); Mesiti (2002), several relevant applications can be devised for such a structural similarity measure. The measure can be applied for the classification of a document D against a set of DTDs in order to identify the DTD structure closer to D. Whenever such a DTD is identified, all policies and data structures defined on the DTD for document storage, access, and protection can be applied to D. The measure can also be applied for the evaluation of schema-based queries over sources of XML documents. A query can be represented as a DTD in which content constraints are specified against data content elements, and the measure is employed for identifying the documents that closely address the query constraints and ranking them relying on the similarity degree. A further application of the measure is for the selective dissemination of XML documents (Bertino, Guerrini, & Mesiti, 2004b; Stanoi, Mihaila, & Padmanabhan, 2003) relying on user profiles. Selective information dissemination is obtained by matching each document in the continuous incoming data stream against the DTD(s) modeling the user profile, and distributing the document to a user if it is similar enough to her profile. A final application of the measure is in the protection of

sources of XML documents (Bertino, Castano, Ferrari, & Mesiti, 2002). Security policies specified on a DTD can be propagated on the parts of a document that match against it. This approach simplifies the work of a security officer that should protect a massive amount of information arriving from external sources.

Measuring this kind of structural similarity is not, however, a trivial task. Many factors should be taken into account in the evaluation, like the hierarchical and complex structure of XML documents and DTDs, and the tags used to label their elements. The measure should thus be developed taking into account several requirements, extensively discussed and motivated in Bertino et al. (2004a).

1. *Maximization of common features*. The measure should give more relevance to the common features between a document and a DTD. Since more than one structure can be generated starting from the same DTD, the measure must choose the structure that, having the same number of common features, minimizes the different features.
2. *Levels at which common and different features are detected*. Common and different features at higher levels in the hierarchical structure of the documents are more relevant than the ones deeply nested in the document structures.
3. *Structure of elements that are not in common between the two structures*. The lack (or the extra presence) of an element with a complex structure should have a higher impact than an element with a simpler structure.
4. *Optional and repeatable elements*. In case of repeatable elements (i.e., elements marked by the * or +), the similarity measure should identify the best number of repetitions, that is, the one that maximizes common features and minimizes different features. Note that a higher number of repetitions can result in every element in the document matching with an element in the DTD but, however, it can increase the unmatched elements in the DTD.
5. *Irrelevance of order.* Taking order into account leads to prefer matches with a lower number of common elements, in the same order, to matches that maximize common elements, but in a different order. This is not acceptable in our application contexts.
6. *Level by level matches.* The measure should determine, level by level, elements that are in common between the document and the DTD and it should not allow matches among elements in different positions in the hierarchical structure. Such kind of matches are not allowed for semantic and efficiency reasons (Wang, DeWitt, & Cai, 2003): the element *context* in an XML document strongly contributes to determine which information that element models.
7. *Tag equality vs. tag similarity.* Since the structural constraints a DTD imposes on the document structure are weakened, it does not make sense to require tag names in the document to exactly coincide with those of the DTD. A document element could be valid with respect to a DTD specification, except for its tag. The two tags can be different but can still represent the same information: they can be synonyms (e.g., *movie* and *film*), or syntactically similar according to a string edit distance function (e.g., *chtitle* and *title*).

Different relative relevance can be assigned to different requirements, that can be set by users depending on the application domain in which the measure is employed. This is realized relying on parameters that state the relevance of divergent elements

with respect to common elements, of a level with respect to the underlying level, of similar tags with respect to equal tags.

Starting from these requirements, in this paper we define and formalize a structural similarity measure between an XML document and a DTD and provide an algorithm for its computation. Note that, being XML documents and DTDs represented as trees, the problem can be seen as an extension of the tree editing problem (Zhang & Shasha, 1989). We indeed compute the (structural) similarity between a tree and an intensional representation of a set of trees (generated from the DTD). This result could be achieved through an *extensional* approach: the set of document structures described by the DTD is made explicit and for each document structure the tree edit distance is computed, choosing the document structure that minimizes the distance. This approach is alternative to our *intensional* approach, which is specialized, offers better performance for unordered trees.

Specifically, the contributions of the paper can be summarized as follows:

- *Formalization of the requirements.* The requirements, emerged in Bertino et al. (2004a) and extended with tag similarity, are formalized here through the definition of a mapping relationship.
- *Formalization of the solution.* The algorithm for computing the similarity measure is formally defined and its complexity is discussed by identifying the conditions—the most common in practice—under which the algorithm is polynomial and by proposing techniques for reducing the execution time in the worst case.
- *Experimental evaluation.* The requirements have been validated and the paper reports the results of this evaluation. Moreover, an experimental validation of the measure is presented that shows the applicability of the measure in practice.

The paper is organized as follows. Section 2 formally states the problem and possible approaches to face it. Section 3 presents the intensional approach for evaluating the structural similarity. Section 4 formally defines the structural similarity measure and its properties. Some experiments performed for assessing the effectiveness and efficiency of our approach are discussed in Section 5. In Section 6 a comparison with related work is provided. Finally, concluding remarks are presented in Section 7. Appendix I contains some details of the matching algorithm, whereas Appendix II includes proof sketches.

2 Problem statement

In this section we formally state the problem of measuring the structural similarity between an XML document and a DTD, addressing the requirements outlined in the Introduction. We first introduce our representation of XML documents and DTDs. Then, we define a structural similarity measure between documents. Relying on such a document to document measure, we state the problem of measuring similarity between a document and a DTD and outline our approach for its solution.

2.1 Documents and DTDs as labeled trees

Documents and DTDs are represented as labeled trees. Our definitions are based on the ones developed in Bertino et al. (2004a). A labeled tree is a tree T associated with

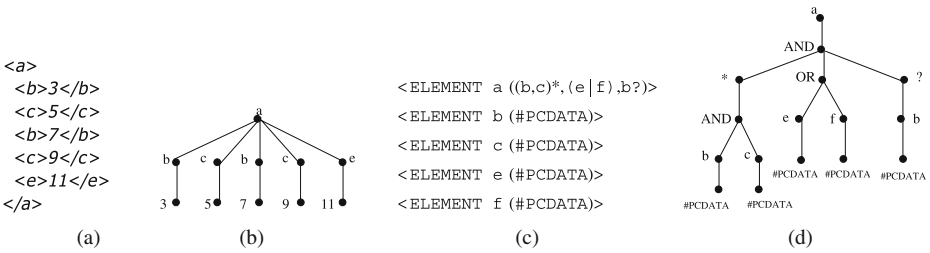


Fig. 1 Examples of XML document and DTD (**a**,**c**) and their tree representations (**b**,**d**)

a node labeling function φ. Let N be a set of nodes and A a set of labels, a labeled tree is inductively defined as follows: $v \in N$ is a labeled tree with $\varphi(v) \in A$; if C is a set of labeled trees ($C = [T_1, \ldots, T_n]$), (v, C) is a labeled tree with $\varphi(v) \in A$.

Attributes are modeled as special data content elements and no longer considered. The labels used for labeling a document are from a set $\mathcal{EN}$ of element tags, and a set $\mathcal{V}$ of `#PCDATA` values. By contrast, the labels used for labeling a DTD are from the set $\mathcal{EN} \cup \mathcal{ET} \cup \mathcal{OP}$, where $\mathcal{ET} = \{$`#PCDATA`, `ANY`$\}$ is the set of basic types for elements, and $\mathcal{OP} = \{$`AND`, `OR`, `?`, `*`, `+`$\}$ is a set of operators. The `EMPTY` type is not considered because it is a special case of the `#PCDATA` type with empty content. The `AND` operator represents a sequence of elements, the `OR` operator represents an alternative of elements (at least one of the alternatives must be selected), the `?` operator represents an optional element, whereas the `*` and `+` operators represent repeatable elements (with the usual meaning). The root label belongs to $\mathcal{EN}$ (it is the name of the main element of documents the DTD describes) and there is a single subtree of the root. Moreover, the out-degree of a node can be greater than one, only if the node tag is `AND` or `OR`.

Examples of an XML document and DTD and their tree representations are shown in Fig. 1. Given a document/DTD tree D and a node x of D, Table 1 reports the notations used for its representation as a labeled tree. We remark that, when x is an element of a DTD, $children(x)$ is the set of subelements of x, independently from the operators used for binding them.

Table 1 Notations

$children(x)$	The subelements of x
$\mathcal{DCE}(x)$	True, if x is a data content element
$root(D)$	The root of D
$l(D)$	The label of the root of D
$l(x)$	The label (or content) of x
$lev_D(x)$	The nesting level of x in D
$n(D)$	The maximal level of element nesting in D
$parent(x)$	The parent of x
$Vertex(D)$	The set of vertices of D
$\|D\|$	Cardinality of $Vertex(D)$
$deg(D)$	Maximal out-degree of D

Example 1 Let D be the document in Fig. 1b and $T = (v, C)$ the DTD in Fig. 1d, with v the root and C the children of T. $l(T) = \mathtt{a}$, $n(T) = 3$, $lev_T(v) = 1$, $|D| = 13$, $deg(D) = 5$.

2.2 Structural similarity between documents

We now present a structural similarity measure between documents that takes into account our requirements. The definition is based on the concept of mapping between the nodes of the tree representation of documents.

2.2.1 Tag similarity

Two tags are similar if they are identical, or they are synonyms relying on a given Thesaurus, or they are *syntactically similar* relying on an edit distance function (Rice, Bunke, & Nartker, 1997).[1]

Definition 2.1 ($\simeq$). Let $l_1, l_2 \in \mathcal{EN}$, $l_1 \simeq l_2$ iff one of the following conditions is verified: (1) $l_1 = l_2$ or (2) l_1 is a synonym of l_2, or (3) l_1 and l_2 are syntactically similar.

Whenever the tags are not identical a penalty should be applied for taking the tag differences into account. In our measure we consider the δ and η parameters ($\delta, \eta \in [0, 1]$) which represent penalties applied when the tags are not identical. We remark that two tags are "more similar" if they are synonyms rather than syntactically similar. Thus, penalty η should be higher than δ.

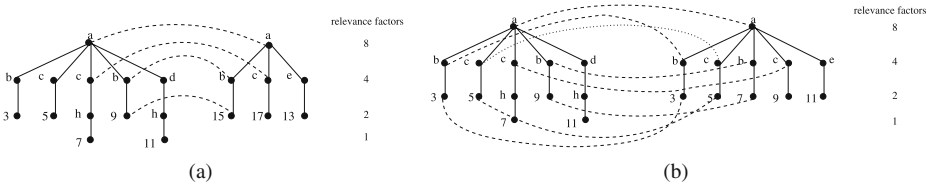
Definition 2.2 (*Tag similarity*) Let $l_1, l_2 \in \mathcal{EN}$. Tag similarity between l_1, l_2 is expressed as:

$$\epsilon(l_1, l_2) = \begin{cases} 1 & \text{if } l_1 = l_2 \\ 1 - \delta & \text{if } l_1 \text{ is a synonym of } l_2 \\ 1 - \eta & \text{if } l_1 \text{ is syntactically similar to } l_2 \\ 0 & \text{otherwise} \end{cases}$$

2.2.2 Mapping among documents

A mapping between a pair of XML documents is a relationship among their vertices that takes documents' tags into account. Moreover, vertices with data content match when the corresponding elements match. Our definition relies on the mapping definition proposed by Nierman and Jagadish (2002). However, our definition differs from the one proposed in Nierman and Jagadish (2002) because we do not require that root labels be identical. In this way we handle the presence of artificial root labels (e.g., `root`, `top`) in the documents. Moreover, tag similarity is allowed.

[1] Two tags are syntactically similar if the string edit function returns a number of edit operations less than a given threshold.



(a) (b)

Fig. 2 Examples of mapping between two XML documents

Definition 2.3 (*Mapping*) Let D_1, D_2 be two documents, and $M \subseteq Vertex(D_1) \times Vertex(D_2)$ be a correspondence between vertices of D_1 and D_2. A triple (D_1, D_2, M) is a mapping from D_1 to D_2, if a set M of pairs of vertices exists such that:

- $(root(D_1), root(D_2)) \in M$;
- For each $(x_1, x_2), (y_1, y_2) \in M$, $x_1 = y_1$ iff $x_2 = y_2$ (one-to-one correspondence);
- If $(x_1, x_2) \in M$, then $(parent(x_1), parent(x_2)) \in M$ and $(l(x_1) \simeq l(x_2)$ or $l(x_1), l(x_2) \in \mathcal{V})$ (prefix closed).

Figure 2 shows two examples of mapping. Note that, vertices with data content match whenever their corresponding elements match. When no ambiguity arises, we use M instead of (D_1, D_2, M). $\pi_i(M)$ denotes the set of vertices extracted from component i ($i \in \{1, 2\}$) of the pairs in M.

2.2.3 Evaluation of a mapping

The mapping allows the identification of a prefix tree of D_1 and D_2, namely the portions of the trees for which a correspondence has been established by M. The vertices in the prefix tree, as well as the vertices for which no correspondence has been established, should be evaluated according to the level at which they are located. A factor γ is introduced for quantifying the relevance of elements of a level with respect to those of the underlying level.[2] For this reason we associate each element x of D_1 (D_2, respectively), with a *relevance factor* $\rho(x)$ s.t. $\rho(x) = \gamma^{max\{n(D_1),n(D_2)\}-lev_{D_1}(x)+1}$ ($\rho(x) = \gamma^{max\{n(D_1),n(D_2)\}-lev_{D_2}(x)+1}$, respectively); the same relevance is given to vertices of D_1 and D_2 at the same level by $max\{n(D_1), n(D_2)\}$.

A mapping M between D_1 and D_2 partitions their vertices as follows:

- The set of *common elements*, that is, the elements in D_1 that match against the ones in D_2 ($\pi_1(M)$ or $\pi_2(M)$). The value $c = \sum_{(x_1,x_2)\in M} \rho(x_1) \cdot \bar{\epsilon}(l(x_1), l(x_2))$ is the relevance of this set. If $l(x_1), l(x_2) \in \mathcal{V}$, $\bar{\epsilon}(l(x_1), l(x_2)) = 1$, otherwise $\bar{\epsilon}(l(x_1), l(x_2)) = \epsilon(l(x_1), l(x_2))$, that is, the tag similarity as specified in Definition 2.2.
- The set of *plus elements*, that is, the elements in D_1 that do not match against elements in D_2 ($Vertex(D_1) \setminus \pi_1(M)$). The relevance of this set is expressed as $p = \sum_{x_1 \notin \pi_1(M)} \rho(x_1)$.

[2] If $\gamma = 1$ all the elements have the same relevance, if $\gamma = 2$ each level has double relevance w.r.t. the underlying level.

– The set of *minus elements*, that is, the elements in D_2 that do not match against elements in D_1 ($Vertex(D_2) \setminus \pi_2(M)$). The relevance of this set is expressed as $m = \sum_{x_2 \notin \pi_2(M)} \rho(x_2)$.

These three values p, m, c are combined through an *evaluation function* that rates the mapping. The evaluation function we choose is function $\mathcal{E}$ (Bertino et al., 2004a) that is based on the *ratio model* (Tversky, 1977). Let $p, m, c, \alpha, \beta \in \mathbb{R}^+ \cup \{0\}$. Function $\mathcal{E}$ is defined as follows:

$$\mathcal{E}(p, m, c) = \begin{cases} 0 & \text{if } (p, m, c) = (0, 0, 0) \\ \dfrac{c}{\alpha p + c + \beta m} & \text{otherwise} \end{cases}$$

This function computes the ratio between the evaluation c of the common elements (i.e., elements in the "intersection" between the two structures) and the evaluation $p + m + c$ of all the elements in the two structures (i.e., elements in the "union" between the two structures). The obtained value is a real number in the range [0,1]. Plus and minus elements are weighted according to α and β parameters. Depending on the values assigned to these parameters, more relevance can be given to plus elements than to minus elements, or vice-versa. For example, if $\alpha = 0$ and $\beta = 1$ plus elements are not taken into account. Therefore, a document with only extra elements with respect to the ones specified in the DTD has a similarity degree equal to 1. In the following example, whenever not specified, we assume $\alpha = \beta = 1$ and $\gamma = 2$.

$\mathcal{E}val(M)$ is defined as $\mathcal{E}(p, m, c)$, where (p, m, c) is the triple of values representing the evaluation of plus, minus, and common elements resulting from the mapping M.

Example 2 Consider the mapping M_1 in Fig. 2a between document D (left hand side) and D_1 (right hand side). Four common elements (evaluated 18), nine plus elements (evaluated 22), and three minus elements (evaluated 8) are identified. The similarity is $\mathcal{E}val(M_1) = \mathcal{E}val(22, 8, 18) = \frac{18}{48} = 0.375$. Consider now the mapping M_2 in Fig. 2b between the same document D and D_2. Eight common elements (evaluated 30), three plus elements (evaluated 10), and five minus elements (evaluated 8) are identified. The similarity is $\mathcal{E}val(M_2) = \mathcal{E}val(10, 8, 30) = \frac{30}{48} = 0.625$. D is thus more similar to D_2 than to D_1.

2.2.4 Structural similarity between documents

Several mappings can be established between a pair of XML documents. Since we are interested in a mapping that maximizes the common elements, we introduce the notion of *maximal prefix mapping*, that is, a mapping in which the number of common elements is maximal.

Definition 2.4 (*Maximal prefix mapping*) Let D_1, D_2 be two documents, and $\mathcal{M}$ be the set of possible mappings between D_1 and D_2. A maximal prefix mapping between D_1 and D_2, $M^{max}(D_1, D_2)$, is a mapping $M^* \in \mathcal{M}$ such that the cardinality of M^* is greater than or equal to the cardinality of any other mapping M in $\mathcal{M}$ (i.e., $|M^*| \geq |M|, \forall\, M \in \mathcal{M}$).

More than one maximal prefix mapping can be identified between two documents. In the evaluation of the structural similarity between two documents, we consider the maximal prefix mapping that maximizes the $\mathcal{E}$ function (i.e., minimizes plus and minus elements). The two mappings in Fig. 2 are maximal.

Definition 2.5 (*Structural similarity between XML documents*). Let D_1, D_2 be two documents, the structural similarity between D_1 and D_2 is evaluated as:

$$\mathcal{S}_{DD}(D_1, D_2) = max\left\{\mathcal{E}val(M^{max}(D_1, D_2))|M^{max}(D_1, D_2) \text{ is a maximal prefix matching}\right\}$$

Proposition 2.6 *Let D_1, D_2 be two documents, $\mathcal{S}_{DD}(D_1, D_2) = \mathcal{S}_{DD}(D_2, D_1)$.*

This proposition directly follows from the definition of $\mathcal{S}_{DD}$ function.

2.3 Problem definition and approaches

In order to introduce the structural similarity measure between a document and a DTD, the set of document structures generated from a DTD must be introduced. The document structures generated from a DTD are documents with no content for textual elements.

Definition 2.7 (*Set of document structures $\mathcal{G}_d$*). Let T be a DTD. $\mathcal{G}_d(T)$ is the set of distinct document structures that can be generated from T. Each $D \in \mathcal{G}_d(T)$ is valid with respect to T.

The problem of measuring the structural similarity between D and T can be formulated as the problem of identifying the highest structural similarity between D and documents in $\mathcal{G}_d(T)$.

Definition 2.8 (*Extensional structural similarity between a document and a DTD*). Let D be a document, T be a DTD, the extensional structural similarity between D and T is defined as:

$$\mathcal{S}^E_{DT}(D, T) = max\{\mathcal{S}_{DD}(D, D')|D' \in \mathcal{G}_d(T)\}$$

Example 3 Given the DTD T in Fig. 1d, Fig. 3a,b,c show document structures D_1, D_2, D_3 belonging to $\mathcal{G}_d(T)$, whereas Fig. 3d shows a document D not conforming to T. D_2 is the structure with the maximal similarity among those generated from T. Therefore, $\mathcal{S}^E_{DT}(D, T) = \mathcal{S}_{DD}(D, D_2) = \mathcal{E}val(10, 8, 22) = 0.625$.

The above definition precisely characterizes the notion of structural similarity. It also provides an approach for its computation (referred to as *extensional approach*) which is based on the generation of document structures. This approach, however, is not efficient because the set $\mathcal{G}_d(T)$ can be infinite when the */+ operators are employed. By taking the document D into account, only a finite subset $\mathcal{G}_d(T, D)$ of $\mathcal{G}_d(T)$ is considered. This follows from the intuition that repeatable elements of a DTD can be repeated until there are elements in the document that can be matched;

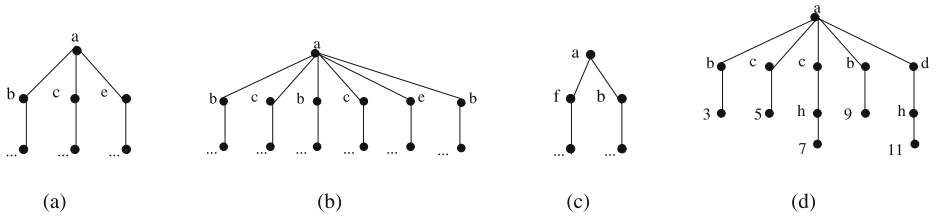


Fig. 3 Structures (**a**,**b**,**c**) are generated from DTD in Fig. 1d, whereas document (d) is not generated from such a DTD

further repeating is useless. The cardinality of $\mathcal{G}_d(T, D)$ is however still exponential in the number of elements of T when the OR, *, and + operators are used.

For this reason we present another approach (referred to as *intensional approach*) in which a $\mathcal{M}atch$ algorithm between the document and the DTD is employed. The DTD is exploited as descriptor of a set of document structures without generating the set of document structures. The grammar rules, specified for constraining the element contents, are exploited for determining the best match. The approach is based on the idea of locally determining the best structure of a DTD element, for elements containing alternatives or repetitions, as soon as the structure of its subelements in the document is known. The $\mathcal{M}atch$ algorithm thus identifies a "best structure" by matching the element structure against the DTD specification.

3 The intensional matching approach

In this section we present the $\mathcal{M}atch$ algorithm, an example of its execution, and, finally, its complexity analysis.

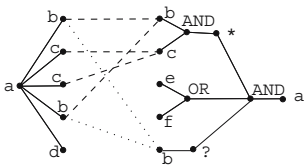
3.1 The matching algorithm

The $\mathcal{M}atch$ algorithm is employed for measuring the structural similarity between a document and a DTD. Since our approach works level-by-level, here we specify

Input: E_d element of the document, E_t DTD element specification
Output: Triple (p, m, c) Var: $R = \emptyset$

1) If (E_t is a basic type definition) return $BasicEval(E_d, l(children(E_t)))$
2) For each $x \in children(E_t)$, for each $y \in children(E_d)$, if $l(x) \simeq l(y)$ $R \leftarrow R \cup \{(x, y, \mathcal{M}atch(x, y))\}$
3) If $(x, y, e_1), (x, z, e_2) \in R$ exist s.t. $y \neq z$,
   identify configurations in R s.t. each node in $\pi_1(R)$ is associated with at most one node in $\pi_2(R)$
4) For each configuration $\bar{R}$ compute $\mathcal{M}(\bar{R}, children(E_t), \mathtt{F})$ and choose the triple (p, m, c) with maximal $\mathcal{E}(p, m, c)$
5) Compute $p' = \sum_{x \in children(E_d), x \notin \pi_1(R)} \mathcal{W}eight(x)$
6) Return $(p + p', m, \rho(E_d) \cdot \epsilon(l(E_d), l(E_t)) + c)$

Fig. 4 Steps of the $\mathcal{M}atch$ algorithm



(a)

b_{1T}	b_{2T}
$\{b_{1D}, b_{2D}\}$	$\emptyset$
$\{b_{1D}\}$	$\{b_{2D}\}$
$\{b_{2D}\}$	$\{b_{1D}\}$
$\emptyset$	$\{b_{1D}, b_{2D}\}$

(b)

Fig. 5 Motivating example for the use of configurations

the behavior for a single element. The application of the algorithm to the roots of the document and the DTD allows the evaluation of similarity between the entire structures.

Figure 4 shows the $\mathcal{M}atch$ algorithm for the root element E_d, of the document in Fig. 3d, and E_t of the DTD in Fig. 1d. The algorithm returns a triple (p, m, c) containing the evaluation of common and different features considering the entire structures of subelements. For the sake of clarity, the subscript D, (respectively, T) is used when we refer to an element of the document, (respectively, DTD). Sibling elements with similar tags are numbered from left to right.

First, the algorithm checks whether E_t is the declaration of a basic type element (`#PCDATA`, `ANY`), or it is the declaration of an element with subelements. In the first case (handled through function *BasicEval* in Fig. 6), if E_t is a `#PCDATA` element and E_d is a data content element, the match is perfect. By contrast, if E_d contains subelements, then the algorithm evaluates the missing of the `#PCDATA` content and the exceeding presence of subelements of E_d. If E_t is an `ANY` type, the subelements of E_d perfectly match this type. In the second case, the one reported in Fig. 5a, the algorithm identifies the subelements of E_d that match (because they have similar labels) the subelements of E_t, independently from the operators used in the declaration of E_t. In Fig. 5a, a dashed line has been drawn between the matching nodes. The identified matches are collected in the set of correspondences R. Moreover, the $\mathcal{M}atch$ algorithm is recursively applied on the pairs of subelements to determine their similarity.

The algorithm then considers the correspondences in R. Each correspondence can associate a subelement of E_d with a single subelement of E_t. By contrast, a correspondence can associate a subelement of E_t with more than a subelement of E_d only when it is declared repeatable (a subelement of E_t is repeatable when the `*`/`+` operator is specified for it or for a `AND`/`OR` group of elements it belongs to). Therefore, in the example, element b^1_T can be associated both with b^1_D and b^2_D, and b^1_D can be associated either with b^1_T or b^2_T. For this reason, the possible subsets of R, named *configurations*, in which each node in $\pi_1(R)$ is associated with at most a node in $\pi_2(R)$ are considered. In the example, the four configurations in Fig. 5b should be

$$BasicEval(T_d, \{l\}) = \begin{cases} (0, 0, \rho(T_d)) & \text{if } l = \texttt{\#PCDATA} \text{ and } \mathcal{DCE}(T_d) = \texttt{T} \\ (\mathcal{W}eight(T_d), \rho(T_d), 0) & \text{if } l = \texttt{\#PCDATA} \text{ and } \mathcal{DCE}(T_d) = \texttt{F} \\ (0, 0, \mathcal{W}eight(T_d)) & \text{if } l = \texttt{ANY} \end{cases}$$

Fig. 6 Function *BasicEval*

considered. They represent the different ways of distributing the two b_D elements against the two b_T elements.

Element declarations in which two subelements have similar tags rarely occur in real DTDs modeling data-centric documents and mainly occur because of a bad DTD design. Indeed, in the declaration of an element, there is no need to specify twice the same subelements (or similar ones). Thus, the evaluation of a single configuration is required in most common situations. However, for sake of completeness, we discuss the general situation.

In Step 4, function $\mathcal{M}$ is applied as many times as the number of configurations. Each recursive call returns the evaluation of the match between the content of E_D and the specification of E_T by taking the operators employed and the distribution of the elements specified in the configuration into account. Among the computed evaluations, the maximal one is then selected as the highest evaluation of similarity between E_D and E_T subelements. In the example, the best match corresponds to the first configuration. Details of the $\mathcal{M}$ function are in Appendix I.

After that, the subelements of E_d not matched against a subelement of E_t are evaluated as plus elements. This is obtained through function *Weight* which returns the value p' representing the relevance of plus subelements. Minus subelements of E_t are evaluated by function $\mathcal{M}$ also employing function *Weight*. In both cases, function *Weight* has the purpose to evaluate the lack (or the extra presence) of an element depending on its structure and its relevance factor. Since plus elements are elements of a document, their evaluation is performed by adding the relevance of their children/descendants. By contrast, since minus elements are DTD specifications, their evaluation is a little more complex. In this case, the simplest structures the DTD can generate for those DTD portions need to be identified. Thus, optional and repeatable elements should not be considered and, in case of alternatives, the alternative with the simplest structure must be chosen. Finally, the relevances of all elements of the structure identified in this way are added. Details on function *Weight* can be found in Appendix I.

Once the evaluation (p, m, c) of plus, minus, and common subelements of E_d and E_t is obtained as specified so far, the evaluation of similarity of E_d and E_t is performed as follows. Common elements between E_d and E_t are evaluated by adding the relevance factor of the root of E_d and the root of E_t to c. Plus elements are evaluated by adding the value p' to p. The evaluation of minus elements is m itself. Note that when the *Match* algorithm is applied on the roots of a document and DTD and the roots do not have similar labels, the relevance factor of the roots is zero. That is, they do not contribute in the evaluation of similarity (artificial roots), whereas the evaluation of their children does.

3.2 Execution example of the *Match* algorithm

Consider the document D in Fig. 3d and the DTD T in Fig. 1d. The *Match* algorithm identifies the subelements of D (i.e., elements tagged `b`, `c`, `d`) and those of T (i.e., elements tagged `b`, `c`, `e`, `f`) that are bound by the $\simeq$ relationships (i.e., elements tagged `b`, `c`). The possible matchings depicted in Fig. 5a are identified. For each matching, the *Match* algorithm is hence invoked and a triple (p, m, c) containing its evaluation is returned. For example, $\mathcal{M}atch(b^1_D, b_T) = (0, 0, 6)$ because the element perfectly matches the basic specification of b_T (i.e., the labels are identical and the

b^1_D content is a textual content); $\mathcal{M}atch(c^2_D, c_T) = (3, 2, 4)$ because the two elements have the same label, but c^2_D has a plus content (evaluated 3 by function $\mathcal{W}eight$) whereas c_T requires a textual content which is missing. In summary, the recursive calls of the $\mathcal{M}atch$ algorithm returns the set $\{(b^1_D, b^1_T, (0, 0, 6)), (b^2_D, b^1_T, (0, 0, 6)), (b^1_D, b^2_T, (0, 0, 6)), (b^2_D, b^2_T, (0, 0, 6)), (c^1_D, c_T, (0, 0, 6)), (c^2_D, c_T, (3, 2, 4))\}$. We remark that, even if two b elements are used in a_T, their structures are always identical and thus they are evaluated only once.

Since the specification of T contains two elements with the same label (i.e., `b`), different configurations are extracted and used in the $\mathcal{M}$ function to determine and evaluate the missing and exceeding subelements of D, by exploiting the matchings depicted in Fig. 5b. Consider, for example, the configuration $\bar{R} = \{(b^1_D, b^1_T, (0, 0, 6)), (b^2_D, b^1_T, (0, 0, 6)), (c^1_D, c_T, (0, 0, 6)), (c^2_D, c_T, (3, 2, 4))\}$ in which the two b_D subelements are associated with b^1_T. The recursive calls of function $\mathcal{M}$ trace the operators that are used in the specification of $\mathtt{a}_T$. By taking the triples generated in $\bar{R}$ into account, on return from the recursive calls, the operators and the repeatability of a node are considered in order to select the best choice among the possible ones for binding together subelements. For what concerns the non-repeatable `AND` operator the following evaluations are performed.

- For the repeatable `AND` operator that binds elements `b` and `c`, function $\mathcal{M}$ composes the triples in the configuration $\bar{R}$ obtained for the underlying elements in order to evaluate the repetitions of the two subelements. Such an evaluation is $(0, 0, 12)$ for the first occurrence of the `AND` operator (obtained from $(b^1_D, b^1_T, (0, 0, 6)) + (c^1_D, c_T, (0, 0, 6))$), and $(3, 2, 10)$ for the second one (obtained from $(b^2_D, b^1_T, (0, 0, 6)) + (c^2_D, c_T, (3, 2, 4))$). The two triples are then summed in the evaluation of the `*` operator (i.e., $(3, 2, 22)$).
- For the `OR` operator that binds two missing subelements, function $\mathcal{M}$ evaluates its lack by choosing the alternative with less impact in the evaluation (i.e., $(0, 6, 0)$).
- For the `?` operator, the lack of matching with b^2_T is evaluated $(0, 0, 0)$ because it is allowed that no elements match against it.

The sum of the so obtained evaluations, i.e., $(3, 8, 22)$, is the evaluation of the non-repeatable `AND` operator that concludes the evaluation of function $\mathcal{M}$ for configuration $\bar{R}$. The other configurations, outlined in Fig. 5b, return triples with a similarity smaller than that of $(3, 8, 22)$. Thus, the algorithm uses this triple to obtain the similarity between D and T. The algorithm evaluates the plus elements ($p' = 7$ for the extra presence of element d_D) and the perfect match of the root element tags (8). Then, the triple $(10, 8, 30)$ is returned by the algorithm. The similarity degree between D and T is thus $30/48 = 0.625$. Note that, this is the same evaluation obtained in Example 2. The structure in Fig. 3b is the most similar to D among the ones generated from T.

3.3 Complexity of the *Match* algorithm

The complexity of the $\mathcal{M}atch$ algorithm strictly depends on function $\mathcal{M}$ and on the number of configurations. Function $\mathcal{W}eight$, for determining the complexity of $\mathcal{M}atch$, can be thought as pre-computed for each node of the document and the DTD and associated with this node (at a cost of $\mathcal{O}(|D| + |T|)$ which does not affect the

overall complexity. We remark that in our context we normally have to deal with a single configuration. However, for the sake of completeness, we specify the maximal number of configurations and discuss some techniques to reduce this number and thus improve the overall execution time.

3.3.1 Complexity of function $\mathcal{M}$

Given a configuration R, representing the correspondences among the subelements of E_d and E_t, the number of operations function $\mathcal{M}$ performs on R depends on the cardinality of R. Indeed, function $\mathcal{M}$ evaluates the operators and combines the evaluations obtained on the common elements between the two structures (contained in the configuration R) with the evaluation of plus and minus elements (obtained by function *Weight*).

Proposition 3.1 Let T_t be a subtree of a DTD specification E_t and τ be the cardinality of the configuration R. The cost $Op(\tau)$ of the operations performed on the τ correspondences depends on the label l of the root of T_t. Specifically: if $l \in \{\texttt{\#PCDATA}, \texttt{ANY}, ?\}$, $Op(\tau) \in \mathcal{O}(1)$; if $l \in \mathcal{EN}$, $Op(\tau) \in \mathcal{O}(\tau \cdot log(\tau))$; if $l \in \{\star, +\}$, $Op(\tau) \in \mathcal{O}(\tau)$; if $l \in \{\texttt{AND}, \texttt{OR}\}$, $Op(\tau) \in \mathcal{O}(\tau^2)$.

From Proposition 3.1, it follows that in the worst case, the cost of the operations performed on a configuration is quadratic in the number of correspondences. Moreover, the number of recursive calls of function $\mathcal{M}$ on the operators used in the DTD is linear in the number of nodes of the two structures. The complexity of function $\mathcal{M}$ is thus the product of these two quantities.

Proposition 3.2 Let Γ be the maximal cardinality of a configuration between an element of a document D and an element of a DTD T. The complexity of $\mathcal{M}$ is $\mathcal{O}(\Gamma^2 \cdot (|D| + |T|))$.

The cardinality of a configuration is limited by the number of subelements of an element of the document D. Therefore, the complexity of $\mathcal{M}$ can also be specified as $\mathcal{O}(deg(D)^2 \cdot (|D| + |T|))$.

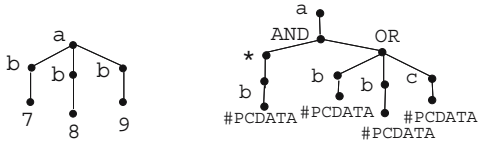
3.3.2 Configurations

If m is the number of subelements with similar labels in E_t, and n is the number of subelements of E_d that are in correspondence with them, the number of configurations that should be checked is m^n, that is, the number of ways the n subelements of E_d can be distributed among the m subelements of E_t.

The number of configurations can be reduced by taking into account that elements with similar labels of E_t can be non repeatable and thus only an element of the document can be associated with them. Moreover, there are configurations that lead to the same similarity degree and, therefore, can be skipped. Finally, it is possible to define some "rewriting rules" that allow one to rewrite a DTD T_1 into another DTD T_2 in which the number of elements with similar labels is reduced. These techniques

can be applied independently. In the discussion we will refer to the document and DTD in Fig. 7a,b. Without applying our techniques, the number of configurations that should be considered is 27. We now show how to reduce this number.

– *Repeatable elements*. Whenever a subelement of E_t is not repeatable, at most one of the subelements of E_d can match it. By contrast, if the subelement is repeatable the number of elements that can be matched against it depends on the other operators appearing in the declaration of E_t. Therefore, once the elements to be matched against the non-repeatable elements are determined, the remaining elements should be distributed among the repeatable elements. Consider, for example, the document and DTD in Fig. 7a,b. Configurations in Fig. 7c should be checked. The number of configurations to be checked decreases from 27 to 8, because the second and third `b` in the DTD are not repeatable.
– *Evaluation of the elements in the document*. The evaluation of the `b` elements of E_d against one of the `b` elements of E_t is already computed and included in the correspondence R. This information can lead to a further optimization. Indeed, if the match of the $\mathtt{b}_D$ elements against the $\mathtt{b}_T$ element generates triples for which function $\mathcal{E}$ returns an identical value, there is no need to consider which `b` is matched against the `b`'s in the DTD, but just their number. The number of configurations decreases by a factor $n!$ because only the number of subelements of E_d assigned to a subelement of E_t is considered. This number is expressed as $\mathcal{Q}_{n,m} = \binom{n+m-1}{m-1}$ that is, the number of ways for distributing n items in m buckets disregarding which item falls in a bucket (i.e., considering only the number of items that fall in a bucket). Consider, for example, the document and the DTD in Fig. 7a,b. Since the evaluations of the $\mathtt{b}_D$ elements against one of the $\mathtt{b}_T$ elements are identical, the number of configurations is $\mathcal{Q}_{3,3} = 10$. Taking also the repeatability of the `b` subelements of `a` into account, the configurations reduces to 4: the first, second, fourth, and last row of Fig. 7c.
• *Normalization of the DTD*. Another technique for reducing the configurations consists in the identification of rules for translating element declarations, having subelements with similar tags, into equivalent ones with less subelements with similar tags. Sometimes the user can also be interested in rules that translate the DTD in a less constraining DTD (details in Bertino, Guerrini, and Mesiti



b_{1T}	b_{2T}	b_{3T}
$\{b_{1D}, b_{2D}, b_{3D}\}$	$\emptyset$	$\emptyset$
$\{b_{2D}, b_{3D}\}$	$\{b_{1D}\}$	$\emptyset$
$\{b_{1D}, b_{3D}\}$	$\{b_{2D}\}$	$\emptyset$
$\{b_{1D}, b_{2D}\}$	$\{b_{3D}\}$	$\emptyset$
$\{b_{2D}, b_{3D}\}$	$\emptyset$	$\{b_{1D}\}$
$\{b_{1D}, b_{3D}\}$	$\emptyset$	$\{b_{2D}\}$
$\{b_{1D}, b_{2D}\}$	$\emptyset$	$\{b_{3D}\}$
$\{b_{1D}\}$	$\{b_{2D}\}$	$\{b_{3D}\}$

(c)

Fig. 7 A document and DTD having subelements with similar tags (**a**,**b**) and the corresponding configuration table (**c**)

(2004c)). A consequence is that non-valid documents for the original DTD could become valid for the rewritten DTD. However, there are two reasons for considering less constrained DTDs. First, the user could be interested in obtaining an answer more quickly, even if this is less accurate. Second, the less constrained DTD can be used as a filter for quickly eliminating documents that are really dissimilar with respect to the DTD. Then, the selected documents can be compared against the original DTD for obtaining the correct similarity degree.

4 Similarity measure

The similarity measure computed exploiting the DTD rules is defined as follows.

Definition 4.1 (*Intensional structural similarity between a document and a DTD*). Let D be a document and T a DTD. The intensional structural similarity between D and T is defined as

$$\mathcal{S}^I_{DT}(D, T) = \mathcal{E}(\mathcal{M}atch(D, T))$$

We now state the relationship between $\mathcal{S}^E_{DT}$ (Definition 2.8) and $\mathcal{S}^I_{DT}$ (Definition 4.1).

Proposition 4.2 Let D be a document and T a DTD. $\mathcal{S}^I_{DT}(D, T) = \mathcal{S}^E_{DT}(D, T)$.

By means of this result, there are two distinct, but equivalent, approaches for evaluating the structural similarity between a document and a DTD. The intensional approach, however, determines the similarity in polynomial time in the most common situations even if the `OR`, `*`, and `+` operators are used. The extensional approach could be employed whenever the number of configurations to be checked is high. As future work we plan to investigate whether it is better to follow the intensional or the extensional approach in presence of DTD declaration with sibling elements with similar tags.

We now state the relationship between validity and the structural similarity measure.

Proposition 4.3 Let D be a document and T a DTD. The following properties hold:

- If D is valid with respect to T, then $\mathcal{S}^I_{DT}(D, T) = 1$;
- If $\mathcal{S}^I_{DT}(D, T) = 1$ then D is valid with respect to T, disregarding the order of elements.

5 Experimental results

The $\mathcal{M}atch$ algorithm has been implemented in Java, using the DOM libraries. Several experiments have been carried out in order to assess the similarity measure and the matching algorithm both from the correctness and the efficiency viewpoints. First, we tested whether the requirements we identified actually correspond to the

"common feeling" of similarity between an XML document and a DTD. Then, we considered both real and synthetic data and classified them against a set of DTDs in order to verify that the algorithm correctly ranks documents according to the similarity measure. Some performance evaluations have been also carried out to show that the algorithm is reasonably efficient to be used in practice. Finally, we checked whether semantic tag similarity can be used for XML documents gathered from the Web.

5.1 Validation of the requirements

In order to verify that the requirements we identified actually correspond to the "common feeling" of similarity between an XML document and a DTD, we asked twenty people to evaluate the similarity between fifteen documents and a DTD. They had to give a mark between 0 and 10 to the similarity of each document with respect to the DTD. The DTD is about reimbursements of business trips. The twenty people that participated in the experiment have different levels of familiarity with XML, ranging from basic knowledge to common use for work, no knowledge of our approach, and different levels of familiarity with the application domain to which the documents refer. Documents range from valid documents to documents that do not correspond at all to the DTD. We do not report here the documents and the DTD, also because they are in Italian; however, we summarize their characteristics with respect to the DTD in Table 2a. Table 2b reports the average evaluation obtained by each document. Documents are ranked based on our similarity measure. The only case in which the evaluation of the matching algorithm is lower than the average mark is document 13. The outermost element tag of this document is different from the one required in the DTD. This document received a really low average mark because it does not have any common element with the DTD. Our algorithm returns 0, which we think it is the right structural similarity because the document deals with a very different topic. We remark that even if the differences between the average mark and the value our algorithm returns is high, it can be mitigated by choosing higher values of α and β. Actually, if we choose $\alpha = 3$ and $\beta = 3$ (instead of $\alpha = 1$ and $\beta = 1$) the standard deviation moves from 3.3 to 1.6.

Some further considerations should be made on the results for document 12. It is perfectly valid except for the outermost element (the document element) which is different. Some people did not realize that the element tag was different, thus they assigned 10 to the document. However, even among those who found out this difference, some assigned a high value (e.g., 7, or 8), whereas others marked it 0. Actually, in such a situation stating that the document is completely different from the DTD is too strong. Our algorithm assigns 8.5 as similarity degree. This value is computed taking into account that the internal structure of the document perfectly matches the DTD specification but the roots are different.

Another important result of this experiment is the validation of the requirements. Indeed, in the 15 documents we have generated, there are groups of documents stressing a particular requirement. For example:

– Documents 4, 10, 3 have minus elements at different levels of the document structure. People ordered them giving more relevance to missing elements at

higher level than missing elements at lower level. Indeed, document 4, having two missing elements at the bottom level, has a higher mark than document 3, which misses an element at the second level.
– Documents 9, 11, 5 have plus elements at different levels of the document structure. As for minus elements, people ranked them according to the level.
– Documents 8 and 11 have the same number of plus elements at the same level, but document 11 has more common elements. Actually, people found that document 11 is more similar than document 8 because it has more "common elements" than document 8.
– Documents 10 and 11 have plus and minus elements at a high level, whereas documents 4 and 9 have plus and minus elements at a low level. Actually, people gave to these pairs of documents a similar evaluation. Therefore, the presence of

Table 2 (**a**) Characteristics of the documents and (**b**) comparison of the similarity degree computed by our algorithm and the average mark.

Doc.	Characteristics
1/2	Valid documents
3	One missing structured element at second level
4	Two missing `#PCDATA` elements at low level
5	Optional structured element repeated three times
6	Wrong binding of two alternative elements (the DTD Requires `(a, b) | (c, d)` but the doc. contains `a, d`)
7	Missing level in the hierarchical structure
8	Two plus `#PCDATA` elements at third level
9	Two plus `#PCDATA` elements at bottom level
10	Two missing `#PCDATA` elements at third level
11	As document 8 with more common elements
12	Valid document, but different outermost element
13	Completely different document
14	Structured element instead of `#PCDATA` element
15	`#PCDATA` element instead of structured element

Doc.	*Match*	Average	Diff
1	10	10	0
2	10	10	0
4	9.9	7.45	2.45
9	9.7	7.74	1.96
11	9.3	7.58	1.72
10	9.3	6.81	2.49
6	9.1	6.64	2.46
8	8.9	7.15	1.75
14	8.6	5.94	2.66
12	8.5	5.53	2.97
5	8.1	6.87	1.23
15	7.9	5.66	2.24
3	7.8	4.88	2.92
7	4.9	3.73	1.17
13	0	0.18	-0.18

extra elements or the absence of the same number of elements at the same level received the same importance.
– Document 3 misses a structured element, whereas document 10 misses two data content elements, one of which at a high level. The evaluation of document 3 is worse, thus the weight of the missing portion of document has been considered.

5.2 Experiments on real and synthetic data

To validate the proposed technique we have performed some experiments over "real data," gathered from the Web, and "synthetic data," randomly generated. In what follows, we briefly report the most interesting experiments. As remarked in Yao, Ozsu, and Keenleyside (2002) we still lack large repositories of data centric XML documents.[3] Thus, also in the case of "real data" we started from HTML documents and we extracted XML documents from them. Data come from two sources of related documents on the Web. These sources contain HTML documents describing software products and their features. By means of structure extraction tools we obtained 345 documents from the first source and 473 documents from the second one. Since there was no common DTD for all the documents, we randomly extracted some small groups of documents and we manually generated the DTDs representing the structure of documents in each group. Each of the generated DTD thus expresses the structure of a subset of documents in the source. We then applied the similarity measure to all the documents in the source for identifying the DTD in which most documents were classified. Synthetic data were generated by means of a random generator we have developed. We generated 10,000 documents containing elements arbitrarily chosen from the Dublin Core element set (DCI). Each element, representing information about a resource (e.g., `title`, `creator`, `subject`), is optional, repeatable, and may appear in any order. Moreover, they can have qualifiers, that is, subelements, refining their meaning. For example, the `creator` element can have the `personalName` subelement specifying the name of the creator. We have then specified three DTDs using subsets of the Dublin Core element set.

In both the experiments, by matching all the documents in the sources against the DTDs, we have obtained that for each document D, and for each pair of DTDs T_1, T_2 such that D is valid neither for T_1 nor for T_2, whenever $\mathcal{S}^I_{DT}(D, T_1) > \mathcal{S}^I_{DT}(D, T_2)$, the document D is more similar to T_1 than to T_2, according to an actual analysis of the document and the DTD. This analysis has been carried out by us with the help of some students, and relies on the requirements identified in the introduction and validated in Section 5.1.

The previous experiments have been carried out by fixing $\alpha = \beta = 1$, $\gamma = 2$. Some additional experiments have been carried out by fixing $\alpha = 0$. In this case, the matching process can be seen as a method to evaluate the structural query that allows

[3] The need of sources of XML documents is also considered by the INEX organization (Fuhr & Lalmas, 2004). Indeed, in 2004, it has identified a new track for the definition of heterogeneous sources of XML documents in order to test new retrieval approaches.

one to retrieve the documents that contain the elements specified in the DTD. The documents with similarity degree equal to 1 are those containing all the elements required by the DTD, and potentially additional elements.

5.3 Performance evaluation

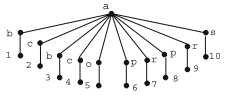
In order to analyze the behavior of our algorithm, we considered the simple document in Fig. 8a and the DTD in Fig. 8b. The DTD is characterized by different combinations of subtrees labeled by operators and of repeatable subtrees. Starting from this document and DTD we defined other documents and DTDs with a higher number of levels by replicating the same structure for each terminal node of the two structures. Figure 8c reports their sizes.

The number of elements in a document depends on the number of levels and varies between 11, for the document in Fig. 8a with a single level, and 11,111, for the document with four levels obtained from document in Fig. 8a. The performance evaluation has been carried out on a Pentium III, 833 Mhz, 512 MB of RAM, with Windows NT as operating system.

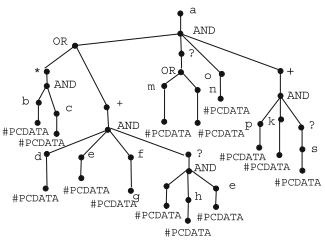
5.3.1 Matching valid documents

Function *Weight* can be computed in different ways. It can be pre-computed with the returned value stored in each node of the DTD/document structure (a-priori); computed *on the fly*, that is, computed when a minus or plus element is found; *materialized*, that is, computed on the fly the first time it is needed and its value stored in the corresponding node of the document (or DTD), and used for further requests.

In order to analyze how these three alternatives affect the performance of the *Match* algorithm, we implemented three versions of the *Match* algorithm. The execution times expressed in milliseconds are reported in the table in Fig. 9a. This table points out three important results. First, even if from the complexity view point pre-computing or materializing the weight does not affect the general complexity of the algorithm, from the point of view of performance there is a big impact. Indeed, when the number of calls to function *Weight* is very low, as in the case of valid documents, the execution time of the a-priori version of the algorithm increases three



(a)



(b)

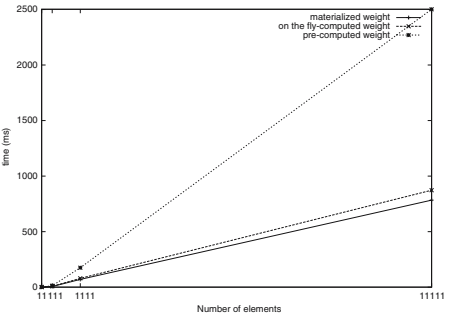
Levels	Doc. size	DTD size
1	208 b	878 b
2	1.79 Kb	15.3 Kb
3	20.3 Kb	297.0 Kb
4	227.0 Kb	5.21 Mb

(c)

Fig. 8 First level of the document (**a**) and DTD, (**b**) for evaluating performance, and (**c**) variation of the size depending on the number of levels

elem.	*Weight*		
	on the fly	a-priori	materialized
11	1.81	1.07	0.66
111	8.00	12.59	6.18
1111	80	175	70.31
11111	873	2502	784

(a)



(b)

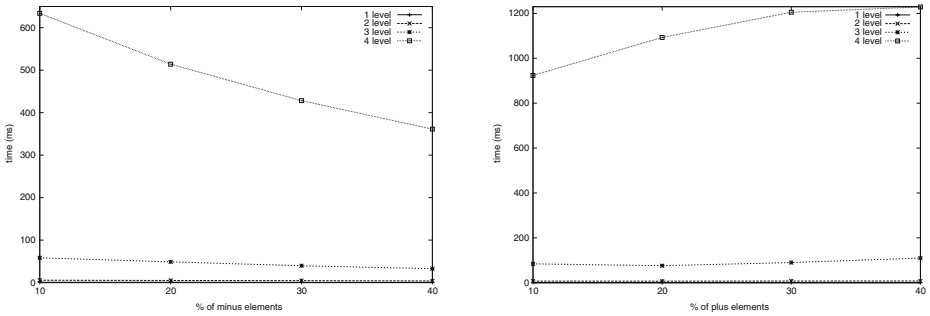
Fig. 9 Performance of the algorithm when matching valid documents

times. By contrast, computing the weight on the fly or materializing it has a little impact on the performance of the algorithm when the document is valid. Actually, it is better to materialize the weight. This last result can also be visualized from the graphs in Fig. 9b which shows that the materialized version of the algorithm is the best. Finally, the execution time of the algorithm linearly grows with respect to the number of nodes of the two structures. Indeed, considering only the number of nodes of the document, the one-level document has 11 elements, whereas, the four-level document has 11,111 elements.

5.3.2 Matching documents with plus and minus elements

In order to analyze the matching algorithm when the document is not valid, we carried out three kinds of experiment.

In the first experiment, we analyzed the influence of missing elements from the document with respect to the DTD. The documents considered in the previous experiments have been progressively pruned of from 10 to 50% their elements. The execution times of the algorithm, expressed in milliseconds and reported in Fig. 10a, decrease as the number of minus elements increases. The execution times are the



(a) (b)

Fig. 10 Performance of the algorithm when matching documents with (**a**) minus and (**b**) plus elements at different levels (materialized version of the algorithm)

result of the execution of the materialized version of the algorithm. However, from the experiments, we observed that the other two versions of the algorithm have the same behavior. Therefore, the presence of minus elements does not increase the execution time of the algorithm, rather it decreases it because in presence of minus elements, no recursive calls of the *Match* algorithm is applied, rather only function *Weight* is called.

In the second experiment, we analyzed the influence of plus elements in the document with respect to the DTD. The documents have been progressively enhanced with additional elements. The new elements have been randomly generated by copying previous elements of the document or by creating new elements. The execution times of the algorithm, expressed in milliseconds and reported in Fig. 10a, progressively increase. Note that for the document with four levels the performance decreases of $\sim$ 35% while the number of nodes increases of 50%.

5.3.3 Matching DTDs with subelements with the same tags

As pointed out in the discussion of the matching algorithm the presence in the DTDs of sibling elements with the same/similar tags changes the complexity of the algorithm from polynomial to exponential. In order to analyze the behavior of the algorithm in this situation, we carried on some more experiments. We considered the DTD in Fig. 7 and the DTD obtained from this DTD by applying the rewriting rules as shown in Section 3.3. Note that the latter DTD does not have subelements with the same tags. Moreover, we considered two valid documents with respect to these DTDs: one having 10 subelements tagged `b`, the other having 20 subelements tagged `b`. For these documents and the DTDs we obtained other documents and DTDs with a higher number of levels. The execution times, expressed in milliseconds, are reported in Table 3. As the reader can see, also for documents with few elements, the execution time of the algorithm is heavily affected. The performance of the enhanced version of the algorithm is, however, good.

5.4 Syntactic vs semantic tag similarity

Exploiting semantic information for the matching of a document with respect to a DTD is an important feature for the similarity measure we developed. Semantic information can be achieved by means of dictionaries, Thesauri, or specific tailored Ontologies. Wordnet (Miller, 1995) is a linguistic Thesaurus that allows to associate each English term with the list of synonyms (synset) that can be used as interpretation of the term.

Table 3 Performance in presence of elements with the same label

Lev.	# of `b` elements			
	10		20	
	Basic	Enhanced	Basic	Enhanced
1	8	0.21	4.55	0.37
2	130	2.32	1,822	8.16
3	12,939	27.66	721,077	179

In order to verify whether the use of Wordnet can be useful for determining the meaning of tags or it is better to exploit the syntactic similarity among tags, we gathered 30 XML documents from the Web and we considered their tags (around 600 distinct tags). It turned out that around 30% of the tags were not present in the Thesaurus because tags were not nouns. Often a tag was a combination of lexemes (e.g., `ProductList`, `SigmodRecord`, `Act_number`), a shortcut (e.g., `coverpg`, `CC`), a single letter word (e.g., `P` for paragraph, `V` for verse), a preposition or a verb (e.g., `from`, `to`, `related`). For the remaining tags a synset has been found in the Thesaurus and was useful to improve the similarity measure in around 20% of the developed tests. These results motivate our decision to consider, in addition to semantic tag similarity, also syntactic tag similarity to cope with the cases in which it is not possible to identify synsets.

These results are quite modest because tags are quite different from textual contents in which Wordnet is usually applied. We are currently working on the development of an approach for learning an Ontology on the basis of tags extracted from a set of documents in the same spirit of Ontology Learning from text approaches (Buitelaar, Cimiano, & Magnini, 2005) but tailored to XML tags.

6 Related work

Approximate matching and structural (and content) similarity for semi-structured data have been investigated at three different levels: (a) among data or between data and a pattern; (b) between data and a schema; (c) among schemas. In the remainder of this section, we first introduce basic approaches dealing with tree similarity, then briefly discuss the main approaches for each of the above levels, and discuss their relationships with our approach. Further details on the approaches presented in this section can be found in Guerrini, Mesiti, and Bertino (2006); Guerrini, Mesiti, and Sanz (2006)

6.1 Tree similarity: Basics

The problem of computing the distance between two trees T_1 and T_2, also known as *tree editing problem*, is the generalization of the problem of computing the distance between two strings (Rice et al. , 1997) to labeled trees. The editing operations available in the tree editing problem are changing, deleting, and inserting a node. To each of these operations a cost is assigned, that depends on the labels of the nodes involved. The problem is to find a sequence of such operations (i.e., an *edit script*) transforming T_1 into T_2 with minimum cost. The distance between T_1 and T_2 is then defined to be the cost of such a sequence. Many approaches generalizing string edit distance to trees have been proposed (Lu, 1979; Selkow, 1977; Tai, 1979; Tanaka & Tanaka, 1988) and the best known result for ordered trees is by Zhang and Shasha (1989). Given two ordered trees T_1 and T_2, their algorithm finds an optimal edit script in time $O(|T_1| \times |T_2| \times min\{depth(T_1), leaves(T_1)\} \times min\{depth(T_2), leaves(T_2)\})$, where $|T|$ denotes the number of nodes in a tree T, $depth(T)$ denotes the depth of a tree T, and $leaves(T)$ denotes the number of leaves

of a tree T. Edit distance for unordered trees has been investigated by Zhang, Statman, and Shasha (1992). Specifically, they have proved that the problem for unordered trees is NP-complete. Zhang (1993) has also proposed a polynomial time algorithm based on a restriction that matching is only allowed between nodes at the same level.

The approaches to tree edit distance measure the (structural and content) similarity between two trees. Our intensional approach, by contrast, as discussed in Section 2.3, computes the (structural) similarity between a tree and an intensional representation of a set of trees (the DTD). Thus, our matching algorithm, while computing the structural similarity, also needs to determine which tree, among those described by the DTD, maximizes the structural similarity. Such a result could be achieved, relying on tree edit distance, through an *extensional* approach: the set of document structures described by the DTD is made explicit and for each document structure the tree edit distance is computed, choosing the document structure that minimizes the distance. This approach is alternative to our intensional approach, which is specialized, offers better performance for unordered trees and addresses our requirements.

Some approaches for approximate tree matching have also been proposed (Zhang, Shasha, & Wang, 1994). Approximate tree matching means matching a tree against a pattern, that is, a tree containing a wildcard (VLDC) that can match with any path/subtree in the other tree. Approximate tree matching with VLDCs is still different (and easier) than the problem we address: a VLDC can match with any path/subtree whereas a DTD is a concise representation for a set of different (but not arbitrary) subtree structures. An element declaration in the DTD, indeed, poses some constraints on the structures allowed for the element in the document.

6.2 Similarity at data level

The approaches along this direction, discussed in Amer-Yahia, Koudas, and Srivastava (2003), can further be distinguished in:

- Approaches for *document structural clustering*, that is, for assembling together documents with a similar structure. This requires to quantify the structural similarity between XML documents, only focusing on document structure, disregarding their content.
- Approaches for *document change detection*, that is, for detecting and representing changes to hierarchically structured information. The changes are detected by comparing the old and new versions of the documents. This approaches do not employ the general tree edit distance algorithm; instead, they develop different algorithms to achieve better performance.
- Approaches for *approximate queries*, that is, for identifying query answers that partially match the query constraints. These approaches share the goal of integrating structural conditions typical of database queries with weighted and ranked approximate answers typical of information retrieval.

Table 4 outlines the characteristics of the most relevant approaches along the three directions. The first column contains the name of the approach, whereas the

Table 4 A comparison among similarity approaches at data level

		Order	Edit Operations	Content	Tag
Structural clustering	Nierman and Jagadish (2002)	Y	Insert, delete, relabel, InsertTree, deleteTree		E
	Flesca et al. (2002)	Y			E
Change detection	Chawathe et al. (1996)	Y	Insert, delete, relabel, MoveTree	E	E
	Chawathe and Garcia-Molina (1997)	N	Insert, delete, relabel, MoveTree, insertTree, deleteTree	E	
	Wang et al. (2003)	N	Insert, delete, relabel	E	
Approximate querying	XIRQL (Fuhr & Grossjohann, 2001)	Y	Relabel	S	E
	XXL (Theobald & Weikum, 2000)	N	Relabel	E	E
	approXQL (Schlieder, 2001)	N	Insert, delete, relabel	S	S

second column states whether the approach considers order among sibling elements. The third column lists the edit operations considered in tree matching. The fourth and fifth columns specify the relationship among data content and tags, respectively, which can be equality (E) or semantic similarity (S).

These approaches measure the similarity between two XML documents; thus their goal is substantially different from ours, which measures the similarity between a tree (the document) and a set of trees, intensionally represented by a DTD. Thus, as discussed above, some of these approaches, specifically those for document clustering, could be adopted to measure the structural similarity between a document and a DTD through an extensional approach.

6.3 Similarity between data and schema

For what concerns similarity between data and schema, existing approaches either address approximate conformance of data to a schema or the extraction of schema information from data relying on their structural similarities.

- *Approximate conformance*: the problem is to detect whether a data instance approximately conform to a schema. The only approaches in this direction are by Grahne and Thomo (2001)and by us, Bertino, Guerrini, Merlo, and Mesiti (1999). None of them, however, has been developed for XML documents and, moreover, in Grahne and Thomo (2001) allowed deviation from the schema are explicitly stated.
- *Schema extraction*: the problem is to discover schema information (in the form of a data guide (Nestorov, Abiteboul, & Motwani, 1998; Wang & Liu, 1998) or of a DTD (Garofalakis, Gionis, Rastogi, Seshadri, & Shim, 2000; Moh, Lim, & Ng, 2000)) from a set of documents. Those approaches do not address how the structure extraction mechanisms can exploit some a-priori knowledge on data schemas. We remark that this knowledge, that we assume in our approach, is often available in practice, for instance when integrating semi-structured data, discovered on the Web, with data having a known structure. Moreover, those approaches work by examining a set of documents at a time, and extracting the schema from these documents. It is not specified whether and how the insertion of new data, once the schema is set, can result in schema modifications. The application of the matching algorithm to the classification of documents we propose is not alternative to those techniques, rather it can complement them. For instance, those techniques can be used to extract, from an initial set of documents, a set of DTDs with respect to which all the documents are then classified by means of our matching algorithm.

Table 5 presents a comparison of the approaches for measuring structural similarity at the data-schema level. The first and second columns report whether the approach has been specifically tailored to XML or it has been developed for semi-structured data (SSD). Then, the representation of data and schema is reported in the third column, together with the main approach followed in solving the problem. The fourth column reports whether the approach considers the order of sibling elements. The fifth column states the relationship holding between data and schema:

Table 5 A comparison among similarity approaches between data and schema

	Data	Schema	Representation/approach	Order	Inclusion	Tag	Rel.
Nestorov et al. (1998)	SSD	Dataguide	Datalog facts and inference rules/ maximal fixpoint semantics	N	Intersection	Equality	F/C
Wang and Liu (1998)	SSD	Dataguide	Tree expression/ frequencies of tree expressions	Both	Contains	Equality + wildcard	F/C
Garofalakis et al. (2000)	XML	DTD	Sequences + regular expressions/ heuristics + MDL principle	Y	Contained	Equality	F/C
Moh et al. (2000)	XML	DTD	Node labeled trees + spanning graph/ frequencies of elements	Y	Contained	Equality	F/C
Grahne and Thomo (2001)	SSD	Dataguide	Edge labeled graph/ transducer	N	Contained	Equality + alternatives	A/D
Bertino et al. (2004a)	XML	DTD	Node labeled trees/ heuristics +association rules	N	Intersection	Equality	F/C
here	XML	DTD	Node labeled tree/ match between trees	N	Intersection	(syn/sem) similarity	F/C

contained means that all the elements appearing in data are also present in the schema, *contains* means that all the elements appearing in the schema are present in data, *intersection* means that data can contain elements that do not appear in the schema and the schema can contain elements not present in data. The relationship among tags, namely, equality, similarity, or equality extended with the possibility to specify alternatives or use wildcards, is specified in the sixth column. Finally, the last column states whether the father-children (*F/C*) relationship is preserved or it is relaxed, i.e., only ancestor-descendant relationship (*A/D*) is preserved.

6.4 Similarity at schema level

Structural similarity at schema level has been extensively investigated in the context of heterogeneous data integration (Batini, Lenzerini, & Navathe, 1986; Parent & Spaccapietra, 1998) and, recently, for DTD clustering (Lee, Yang, Hsu, & Yang, 2002).

In heterogeneous data integration, corresponding components in different schemas need to be identified, keeping both the names and the structures of schema elements into account. In automatic schema matching existing solutions can be differentiated in schema-and-instance level, element-and-structure level, and language-and-constraint-based matching approaches (Do, Melnik, & Rahm, 2003; Rahm & Bernstein, 2001). Prototypes of data integration systems supporting XML schema matching include Cupid, LSD, and COMA. Cupid (Madhavan, Bernstein, & Rahm, 2001) considers both tag names and the hierarchical structure of schemas. Similarity between elements relies on the similarity of their components, thus the name and data type similarities of leaf elements is emphasized. LSD (Learning Source Description) (Doan, Domingos, & Halevy, 2001) is based on machine learning techniques and requires a *training phase* which can incur a substantial manual effort. COMA (Do & Rahm, 2002) provides an extensible library for the application of different approaches and supports various ways for combining matching results.

Despite their differences, those approaches map the original schema (expressed as a DTD or an XML Schema) into an internal schema, more similar to a data guide for semi-structured data than to a DTD. Thus, in performing the structural match, constraints on the occurrences of an element or group of elements are not considered. In our matching algorithm, by contrast, optional and repeatable elements as well as alternative of elements are considered. Moreover, the matching algorithm we propose considers the match between a value (in the document) and a type (in the DTD) and the presence of `ANY` and `EMPTY` types.

In Lee et al. (2002) *XClust*, an integration strategy that involves the clustering of DTDs, is presented. A matching algorithm based on the semantic, immediate descendent, and leaf-context similarity of DTD elements is developed. The algorithm tries to identify possible matches among direct subelements by examining each single element, and considers the element cardinalities (that is, optionality and repeatability) and the similarity of their tags. The internal representation of DTDs is more sophisticated than the ones adopted by the integration systems discussed above. However, DTDs specifying alternative elements are not considered, whereas they are handled by the matching algorithm we propose.

7 Conclusions

Starting from the lack of a structural similarity measure for computing the similarity between a document and a schema, expressed as a DTD, in this paper we have defined such a measure and proposed a matching algorithm to compute it. The proposed algorithm is polynomial in significant cases, very common in practice, and it substantially improves over the extensional approach of evaluating the structural similarity among the document and the document structures intensionally described by the DTD. In the paper, we formally defined the structural similarity measure, presented the algorithm, discussed its time complexity, and presented experimental results that allowed us both to validate the requirements we started with and to show the practical effectiveness of the approach. The proposed measure relies on the requirements of "irrelevance of order" and "level-by-level" matches. These requirements have been motivated in the paper. However, order can be integrated in the measure and considered as a weak constraint in measuring structural similarity. That is, among the matches that maximize common elements we can privilege the ordered one. For space constraints we do not report here how order can be handled, rather we refer to Bertino et al. (2004c) that shows how this weak order constraint can be integrated in our similarity measure. Note, however, that order, as a weak constraint, does not reduce the complexity of the algorithm. As for order also matches at different levels can be integrated in our measure. However, for space constraints, we refer to Bertino et al. (2004c) for its treatment.

We plan to extend the work presented in this paper along several directions. A first direction concerns the use of some "domain dependent" information where the measure is employed. In this way, it is possible to consider in the evaluation of similarity whether two tags, even if not similar (relying on our definition) are interchangeable (e.g., `zip` and `PostalCode`), or two different structures can be used for describing the same kind of data (e.g., the `date` element can be a `#PCDATA` element or contains the subelements `day`, `month`, and `year`). A second direction concerns the adoption of XML Schema instead of DTDs. In this shift, the major challenge concerns the possibility XML Schema gives to define hierarchies of types. A third direction concerns the development of an approach for the optimization of the matching phase between a document and a set of DTDs. By considering some properties of the DTDs it is possible to quickly identify the DTDs that will return an high structural similarity with respect to the documents, without applying the $\mathcal{M}atch$ algorithm against all the DTDs.

Appendix I: Matching algorithm

In this section we present the main characteristics of the $\mathcal{W}eight$ and $\mathcal{M}$ functions of our matching algorithm.

A Function $\mathcal{W}eight$

In functions $\mathcal{M}atch$ and $\mathcal{M}$ a single function, called $\mathcal{W}eight$, has been defined for evaluating the relevance of plus and minus elements starting from the relevance

factor of each node Bertino et al. (2004a). Let T be a subtree of a document or a DTD D, function $\mathcal{W}eight$ is defined as:

$$\mathcal{W}eight(T) = \begin{cases} \rho(T) & \text{if } l(T) = \mathcal{V} \cup \mathcal{ET} \\ \mathcal{W}eight(T') & \text{if } l(T) = + \text{ and } T = (v, [T']) \\ 0 & \text{if } l(T) \in \{\texttt{*}, \texttt{?}\} \\ \sum_{i=1}^{n} \mathcal{W}eight(T_i) & \text{if } l(T) = \texttt{AND} \text{ and } T = (v, [T_1, \ldots, T_n]) \\ min_{i=1}^{n} \mathcal{W}eight(T_i) & \text{if } l(T) = \texttt{OR} \text{ and } T = (v, [T_1, \ldots, T_n]) \\ \rho(v) + \sum_{i=1}^{n} \mathcal{W}eight(T_i) & \text{otherwise, where } T = (v, [T_1, \ldots, T_n]) \end{cases}$$

B Function $\mathcal{M}$

Function $\mathcal{M}$ evaluates the operators employed in the specification of E_t in order to identify the best match between the subelements of E_d and the element specification E_t. This means that for each subelement (or group of subelements) bound by:

- The * (or +) operator: the function determines the right number of repetitions;
- the ? operator: the function determines whether it is better to consider the optional element (or group of elements) present or absent;
- the OR operator: the function determines the best alternative among the possible ones;
- the AND operator: the function evaluates the elements that form the sequence.

Function $\mathcal{M}$ evaluates the operators employed in the specification of E_t in order to identify the best match between the subelements of E_d and the element specification E_t. Function $\mathcal{M}$ relies on the (recursive) evaluations obtained from the configuration R identified among the subelements of E_d and E_t. The parameters of function $\mathcal{M}$ are: the configuration R identified at step 3 of the $\mathcal{M}atch$ algorithm, a subtree T_t of E_t, and a flag r indicating the repeatability of T. If a node is tagged */+, then the direct subelements that it leads are considered repeatable (i.e., several elements of E_d are allowed to match against it). The repeatability flag is initially set to F and it switches to T when a * or + operator is used in E_t in the description of the allowed content of E_d.

In the general case, Function $\mathcal{M}$ returns lists of triples (p, m, c). Each triple in the list is the evaluation of a possible configuration between the subelements of E_d and E_t. In other words, each triple represents the evaluation of a repetition of the operator/subelement of E_t against subelements of E_d. In the following we detail function $\mathcal{M}$.

Base cases

Base cases (whose formal description is reported in Fig. 11) arise when the label l of T_t is an operator but none of the elements it binds corresponds to a child of E_d (formally $\forall x \in children(T_t), x \notin \pi_2(R)$) or l is an element name but none of the children of E_d is in correspondence with it (formally, $T_t \notin \pi_2(R)$). The

$$\mathcal{M}(R, \{T_t\}, r) = \begin{cases} [(0, \mathcal{W}eight(T_t), 0)] & \text{if } l(T_t) \in \{\texttt{OR}, \texttt{AND}, +\}, \text{ and } \forall x \in children(T_t), x \notin \pi_2(R) \\ [(0,0,0)] & \text{if } l(T_t) \in \{^\star, ?\} \text{ and } \forall x \in children(T_t), x \notin \pi_2(R) \\ [(0, \mathcal{W}eight(T_t), 0)] & \text{if } l(T_t) \in \mathcal{EN} \text{ and } T_t \notin \pi_2(R) \end{cases}$$

Fig. 11 Function $\mathcal{M}$: base cases

returned triple represents the evaluation of the lack of T_t in E_d. If the label requires at least an occurrence of T_t ($l \in \{\texttt{OR}, \texttt{AND}, +\} \cup \mathcal{EN}$), then the returned triple is $(0, \mathcal{W}eight(T_t), 0)$, denoting that no common and plus elements have been detected between the subelements of E_d and T_t, and T_t is missing in E_d. By contrast, if the operator states that T_t is optional ($l \in \{^\star, ?\}$), then the triple $(0, 0, 0)$ is returned (the absence of T_t is allowed).

Element tag

When T_t is labeled by a tag, the algorithm checks in R the subelements of E_d that correspond to T_t. The evaluations, previously computed, are employed for determining the number of repetitions of the element (this case is formally reported in Fig. 12). Depending on the repeatability of T_t, the list of triples to return is determined. If $r = \texttt{F}$ the DTD requires just one of the matched elements, whereas the others are plus elements. Therefore, the best triple, among the detected ones, is selected. The evaluation of the exceeding elements is obtained by adding the plus and common components of the remaining triples to the plus component of the selected triple. If $r = \texttt{T}$, by contrast, the best number of repetitions cannot be determined at this level of the structure because it depends on the $^\star/+$ operator that marked the subtree repeatable. Therefore, the function returns a list of triples ordered relying on function $\mathcal{E}$.

$^\star/+$ Operator

When T_t is labeled by the * or the $+$ operator, the evaluation of the match is the value returned by the application of function $\mathcal{M}$ to (the only) child T'_t of T_t, with the repeatability parameter set to $\texttt{T}$ (this case is formally reported in Fig. 13). The presence of the $^\star/+$ operator results in setting the repeatability of the subtree. In this way function $\mathcal{M}$ allows more than one match for the elements the operator binds. The meaning of the list of triples depends on the label l' of T'_t. If $l' \in \mathcal{EN}$ each triple is the evaluation of an occurrence of element l' as subelement of E_d, whereas if $l' \in \{\texttt{AND}, \texttt{OR}\}$ the i^{th} triple in the list is the evaluation of i repetitions of T'_t. The triples in the returned list are added if $l' \in \mathcal{EN}$. In this way we obtain the evaluation of the k subelements tagged l' in E_d, where k is the cardinality of the list. By contrast, if

$$\mathcal{M}(R, \{T_t\}, r) = \begin{cases} [(\sum_{i=1}^{k}(p_i + c_i) - c, m, c)] & \text{if } r = \texttt{F} \text{ and } (p, m, c) = max_{\mathcal{E}}[(p_1, m_1, c_1), \\ & \quad \ldots, [(p_k, m_k, c_k)] \\ Sort_{\mathcal{E}}([(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)]) & \text{if } r = \texttt{T} \end{cases}$$

where, $l(T_t) \in \mathcal{EN}$ and $\{(T_d^1, T_t, (p_1, m_1, c_1)) \ldots (T_d^k, T_t, (p_k, m_k, c_k)\} \subseteq R$.

Fig. 12 Function $\mathcal{M}$: case label is an element tag

$$\mathcal{M}(R, \{T_t\}, r) = \begin{cases} [(\Sigma_{i=1}^{k} p_i, \Sigma_{i=1}^{k} m_i, \Sigma_{i=1}^{k} c_i)] & \text{if } l' \in \mathcal{EN} \\ max_{\mathcal{E}}[(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)] & \text{if } l' \notin \mathcal{EN}, l = */+ \end{cases}$$

where: $l(T_t) = l = {}^*/+$, $children(T_t) = \{T'_t\}, l(T'_t) = l'$ and $[(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)] = \mathcal{M}(R, \{T'_t\}, \texttt{T})$

Fig. 13 Function $\mathcal{M}$: case label is a repeatable (*/+) tag

$l' \in \{\texttt{AND}, \texttt{OR}\}$, a single triple is returned, which is the triple corresponding to the best number of repetitions, that is, the number of repetitions that maximizes function $\mathcal{E}$.

? *operator*

The ? operator is handled as the * operator, except for forcing the maximal number of repetitions to one. Function $\mathcal{M}$ is applied to (the only) child T'_t of T_t, with the r and R parameters unchanged. The application of function $\mathcal{M}$ could return just a single triple (p, m, c) or a list of triples depending on the value of r. If $r = \texttt{F}$ the triple (p, m, c) is returned. If $r = \texttt{T}$ the list is just returned and the evaluation is postponed because it depends on the */+ operator that marked the subtree repeatable.

AND/OR *operator*

The AND and OR operators are the only ones that allow more than one subtree as children. In order to match the children of E_d with one of these operators, we simply consider the lists of triples returned by the application of function $\mathcal{M}$ to each of the children of T_t, leaving unchanged the r and R parameters (these cases are in Fig. 14). If $r = \texttt{F}$, each application of function $\mathcal{M}$ returns a single triple. All the triples are added if the operator is AND. Each triple represents the independent evaluation of a subelement of E_d, their sum evaluates that they appear together. By contrast, if the operator is OR, just the best triple is chosen, whereas the other ones contribute to the evaluation of plus elements. If $r = \texttt{T}$, depending on the operator, a list of triples, containing the evaluation of the possible repetitions of the subtree ordered according to $\mathcal{E}$, needs to be computed. This is obtained through function $\mathcal{H}_{\texttt{AND}}$ or function $\mathcal{H}_{\texttt{OR}}$, reported in the appendix for space limitation. The choice of the best number of repetitions is postponed to the evaluation of the */+ operator that marked the AND/OR operator repeatable.

$$\mathcal{M}(R, \{T_t\}, r) = \begin{cases} \mathcal{H}_{\texttt{AND/OR}}([(T^t_1, \mathcal{M}(R, \{T^t_1\}, r)), \ldots, (T^t_n, \mathcal{M}(R, \{T^t_n\}, r))]) & \text{if } r = \texttt{T} \\ [(\Sigma_{i=1}^{n} p_i, \Sigma_{i=1}^{n} m_i, \Sigma_{i=1}^{n} c_i)] & \text{if } r = \texttt{F} \text{ and } l(T_t) = \texttt{AND} \\ [(\sum_{i=1}^{n} (p_i + c_i) - c, m, c)] & \text{if } r = \texttt{F} \text{ and } l(T_t) = \texttt{OR} \text{ and } (p, m, c) \\ & = max_{\mathcal{E}}[(p_1, m_1, c_1), \ldots, (p_n, m_n, c_n)] \end{cases}$$

where: $children(T_t) = \{T^t_1 \ldots T^t_n\}$ and (in the second and third case) $[(p_i, m_i, c_i)] = \mathcal{M}(R, \{T^t_i\}, r)$, $1 \le i \le n$

Fig. 14 Function $\mathcal{M}$: case label is the AND/OR tag

Figures 15 and 16 contain functions $\mathcal{H}_{\texttt{AND}}$ and $\mathcal{H}_{\texttt{OR}}$, respectively. In the remainder of the section we present the behavior of function $\mathcal{H}_{\texttt{OR}}$ because it is more complex, and an illustrative example.

Function $\mathcal{H}_{\texttt{OR}}$ performs the following operations. First, it determines the *plus* value, summing components of the triples of the lists $r_1, \ldots, r_n$. The components to be added depend on the label of the root of each T_i, $1 \leq i \leq n$. The *plus* value represents the weight of the subtree in case none of the alternatives of the OR operator is chosen. For determining the evaluation of one repetition of the OR operator the triples in the first position in the lists $r_1, \ldots, r_n$ are selected. Among them, the triple (p, m, c), which is the maximum according to the $\mathcal{E}$ function, is extracted (let say it belongs to list r_m, $1 \leq m \leq m$). This way to select the triple ensures that we have identified the subtree T^t_m of the OR that had the best match among the subelements of T_d, according to the $\mathcal{E}$ function. Therefore, among the possible alternatives of the OR operator, we choose the one corresponding to the T^t_m subtree as first repetition of the OR operator.

Function $\mathcal{H}_{\texttt{AND}}(r : list\_of(T : tree, \bar{r} : list\_of(\mathcal{PMC}))) : list\_of(\mathcal{PMC})$
begin */* r is of the form* $[(T_1, r_1), \ldots, (T_n, r_n)]$ **/*
  $rp_{max} = max_{i=1}^{n} \#r_i$;
  for $i = 1$ **to** rp_{max} **do**
  begin
    $(p, m, c) = (0, 0, 0)$;
    for $j = 1$ **to** n **do**
      begin
        Let $r_j = \left[\left(p_1^j, m_1^j, c_1^j\right), \ldots, \left(p_k^j, m_k^j, c_k^j\right)\right]$, l_j the label of the root of T_j and $k = \#(r_j)$;
        If $\left(c_1^j = 0 \textbf{ and } l_j \notin \{*, ?\}\right)$ $(p, m, c) = (p, m, c) + \left(0, i * m_1^j, 0\right)$;
        else case of (l_j)
          $(l_j \in \{\texttt{AND}, \texttt{OR}\})$:
            if $(k \geq i)$ $(p, m, c) = (p, m, c) + \left(p_i^j, m_i^j, c_i^j\right)$;
            else $(p, m, c) = (p, m, c) + (0, (i - k) * \mathcal{W}eight(T_j), 0) + last(r_j)$;
          $(l_j \in \{*, +\})$:
            $(p, m, c) = (p, m, c) + \left(p_1^j, m_1^j, c_1^j\right)$;
          $(l_j = ?)$: */* T_j of the form $(v, [T'_j])$ and l'_j the label of the root of T'_j */*
            if $(l'_j \in \{\texttt{AND}, \texttt{OR}\})$
              if $(k \geq i)$ $(p, m, c) = (p, m, c) + \left(p_i^j, m_i^j, c_i^j\right)$;
              else $(p, m, c) = (p, m, c) + last(r_j)$;
            else
              if $(k \geq i)$ $(p, m, c) = (p, m, c) + \sum_{h=1}^{i} \left(p_h^j, m_h^j, c_h^j\right) + \left(\sum_{h=i+1}^{k} \left(p_h^j + c_h^j\right), 0, 0\right)$;
              else $(p, m, c) = (p, m, c) + \sum_{i=1}^{k} \left(p_i^j, m_i^j, c_i^j\right)$;
          $(l_j \in \mathcal{EN})$:
            if $(k \geq i)$ $(p, m, c) = (p, m, c) + \sum_{h=1}^{i} \left(p_h^j, m_h^j, c_h^j\right) + \left(\sum_{h=i+1}^{k} \left(p_h^j + c_h^j\right), 0, 0\right)$;
            else $(p, m, c) = (p, m, c) + (0, (i - k) * \mathcal{W}eight(T_j), 0) + \sum_{i=1}^{k} \left(p_i^j, m_i^j, c_i^j\right)$;
        end case
      end
    if $(i = 1)$ $list = [(p, m, c)]$; **else** $list = append(list, (p, m, c))$;
  end
  return *list*;
end

Fig. 15 Function $\mathcal{H}_{\texttt{AND}}$

Function $\mathcal{H}_{\texttt{OR}}(r : list\_of(T : tree, \bar{r} : list\_of(\mathcal{PMC}))) : list\_of(\mathcal{PMC})$
begin /* *r is of the form* $([(T_1, r_1), \ldots, (T_n, r_n)])$ *and* l_j *is* $l(T_j)$ */
  $rp_{max} = \sum_{i=1}^{n} \#r_i$;
  $plus = 0$;
  for $j = 1$ **to** n **do**
  begin
    Let $r_j = \left[\left(p_1^j, m_1^j, c_1^j\right), \ldots, \left(p_k^j, m_k^j, c_k^j\right)\right]$ and $k = \#(r_j)$;
    if $(l_j \in \{\texttt{AND}, \texttt{OR}\})$ **begin**
      $r_j = Normalize(r_j)$;
      $plus = plus + \left(p_1^j + c_1^j\right)$;
    end
    else $plus = plus + \sum_{h=1}^{k} \left(p_h^j + c_h^j\right)$;
  end
  for $i = 1$ **to** rp_{max} **do**
  begin
    Let $\bar{r} \in \{r_1, \ldots, r_n\}$ be the list s.t. $\bar{r}[1] = (p, m, c) = max_{\mathcal{E}}([r_1[1], \ldots, r_n[1]])$;
    $deleteHead(\bar{r})$;
    If $(i = 1)$ $list = [(plus - c, m, c)]$; **else** $list = append(list, last(list) + (-c, m, c))$;
  end
  return *list*;
end

Fig. 16 Function $\mathcal{H}_{\texttt{OR}}$

The evaluation $(plus - c, m, c)$ represents one repetition of the OR operator and it is obtained by summing the $(plus, 0, 0)$ triple (which represents zero repetitions of the OR operator) to the triple $(-c, m, c)$. In this way we remove from the plus component the presence of T_m^t and we consider the minus and common element matched with T_d. The list *list* is set to $[(plus - c, m, c)]$. Then, the triple (p, m, c) is removed from r_m because we have to consider the remaining triples for determining the evaluation of the other repetitions of the OR operator. This process is performed until all the $r_1, \ldots, r_n$ lists become empty and *list* contains the triples representing all the possible match found between the OR operator and the subelements of T_d.

Appendix II: Proof sketches

Proof (*of Proposition 3.1*) If $l \in \{\#\texttt{PCDATA}, \texttt{ANY}, ?\}$ the function runs in constant time. If $l \in \mathcal{EN}$ the operation with the highest complexity is the sort of the triples based on the $\mathcal{E}$ function. Since function $\mathcal{E}$ is computed in constant-time, the complexity depends on the number of configurations (in the worst case $\mathcal{O}(\tau \cdot log(\tau))$ using a merge sort). If $l \in \{*, +\}$, the operation with the highest complexity is the computation of the triple that maximizes the $\mathcal{E}$ function. Since function $\mathcal{E}$ is computed in constant-time, the complexity of the operation depends on the size of the list of triples (in the worst case $\mathcal{O}(\tau)$). If $l \in \{\texttt{AND}, \texttt{OR}\}$, the operation with the highest complexity is the computation of function $\mathcal{H}_{\texttt{OR}}$. The complexity of $\mathcal{H}_{\texttt{OR}}$ is $\mathcal{O}(\tau^2)$, because the number of triples it generates is bounded by τ and the second `for`-clause in the function computes τ times the maximum of the lists of triples in input (i.e., $\mathcal{O}(\tau^2)$ operations). Therefore, the complexity of the operation is $\mathcal{O}(\tau^2)$. One may think that the complexity of $\mathcal{H}_{\texttt{AND}}$ is higher than that of $\mathcal{H}_{\texttt{OR}}$. However, this is not true for two reasons. The first reason is that in the computation of the maximum

number of repetitions, function $\mathcal{H}_{\text{AND}}$ considers the size of the list of triples with the highest size, whereas function $\mathcal{H}_{\text{OR}}$ sums the size of all the lists. The second reason is that even if the computations of sum of triples, and sum of plus and common components performed within two nested `for`-clauses, they can be pre-computed, thus decreasing the complexity from $\mathcal{O}(\tau^3)$ to $\mathcal{O}(\tau^2)$. □

Proof (*of Proposition 3.2*) >From Proposition 3.1 we have that for each recursive call the maximum number of performed operations depends on the number of triples τ and is $\mathcal{O}(\tau^2)$. Since Γ is the maximum τ (i.e., the maximal number of configurations that can be established between an element of the document and an element of the DTD), we can conclude that the maximum number of operations on triples is $\mathcal{O}(\Gamma^2)$. Combining together these results we obtain that the complexity of function $\mathcal{M}$ is $\mathcal{O}(\Gamma^2 \cdot (N + M))$. □

Proof (*of Proposition 4.2*) In the evaluation of $\mathcal{S}^E_{DT}(D, T)$ among the document structures generated from T those that have the highest number of common elements are identified. Then, among them the structures that minimize plus and minus elements are chosen. This task is performed starting from the root and moving down level by level to the leaves. The choice of the minus/plus elements depends on the evaluation function $\mathcal{E}$.

In the evaluation of $\mathcal{S}^I_{DT}(D, T)$ the set of document structures is not generated but the same process is performed. For each element, the common subelements are first identified and their structural similarity is (recursively) evaluated. Then, the operators that bind together the subelements are evaluated. This operation can lead a common element to become a plus or minus element (because of exceeding or lacking repetitions of the *⁄+ operators). However, this choice is performed on the basis of the $\mathcal{E}$ function (as done for the evaluation of $\mathcal{S}^E_{DT}(D, T)$ in the generated document structures). □

Proof (*of Proposition 4.3*) The first assertion follows from the fact that if a document D is valid for a DTD T, this means that its structure is exactly one of the structures described by the DTD. Thus, the document neither contains elements not appearing in the DTD (thus, plus = 0), nor it misses elements the DTD requires (thus, minus = 0). Therefore, when function $\mathcal{E}$ is applied, the ratio between c and $0 + 0 + c$ is computed, thus obtaining 1. The second assertion holds since the similarity value can be 1 only if the two values of which we compute the ratio are equal. Since α and β are not null, and the p, m, c values are natural, thus, non negative, this can happen only if $p = m = 0$. This means that the document neither contains elements not appearing in the DTD, nor it misses elements the DTD requires. Thus, according to the notion of validity, if we disregard the order of elements, the document is valid for the DTD. □

References

Amer-Yahia, S., Koudas, N., & Srivastava, D. (2003). Approximate matching in XML. In *Proceedings of the International Conference on Data Engineering* (p. 803). Los Alamitos, California: IEEE Computer Society.

Batini, C., Lenzerini, M., & Navathe, S. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, *18*(4), 323–364.

Bertino, E., Castano, S., Ferrari, E., & Mesiti, M. (2002). Protection and administration of XML data sources. *Data and Knowledge Engineering*, *43*(3), 237–260.

Bertino, E., Guerrini, G., Merlo, I., & Mesiti, M. (1999). An approach to classify semi-structured objects. In *Proceedings of European Conference on Object-Oriented Programming, LNCS (1628)* (pp. 416–440). Berlin Heidelberg New York: Springer.

Bertino, E., Guerrini, G., & Mesiti, M. (2004a). A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications. *Information Systems*, *29*(1), 23-46.

Bertino, E., Guerrini, G., & Mesiti, M. (2004b). Di$\mathcal{X}$eminator: a profile-based selective dissemination system for XML documents. In *Proceedings of the International EDBT Workshop on Clustering Information on the Web*, (pp. 47–54). Berlin Heidelberg New York: Springer.

Bertino, E., Guerrini, G., & Mesiti, M. (2004c). Measuring the structural similarity among XML documents and DTDs. Technical Report, University of Genova. http://www.disi.unige.it/person/MesitiM/publications.html.

Bourret, R. (1999). XML and databases. http://www.rpbourret.com/xml/.

Buitelaar, P., Cimiano, P., & Magnini, B. (2005). Ontology learning from text: methods, evaluation and applications. *Frontiers in Artificial Intelligence and Applications Series*, (pp. 3–12). Amsterdam, The Netherlands: IOS Press.

Chawathe, S. S., & Garcia-Molina, H. (1997). Meaningful change detection in structured data. In *Proceedings of the International Conference on Management of Data* (pp. 26–37). New York: ACM.

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., & Widom, J. (1996). Change detection in hierarchically structured information. In *Proceedings of the International Conference on Management of Data* (pp. 493–504). New York: ACM.

Chinenyanga, T., & Kushmerick, N. (2002). An expressive and efficient language for XML information retrieval. *JASIST*, *53*(6), 438–453.

Cobena, G., Abiteboul, S., & Marian, A. (2002). Detecting changes in XML documents. In *Proceedings of the International Conference on Data Engeneering* (pp. 41–52). Los Alamitos, California: IEEE Computer Society.

DCI. Dublin Core, http://dublincore.org/.

Deutsch, A., Fernandez, M., & Suciu, D. (1999). Storing semistructured data with STORED. In *Proceedings of the International Conference on Management of Data* (pp. 431–442). New York: ACM.

Do, H.-H., Melnik, S., & Rahm, E. (2003). Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, vol. 2593 of *LNCS* (pp. 221–237). Berlin Heidelberg New York: Springer.

Do, H.-H., & Rahm, E. (2002). COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases* (pp. 610–621). San Mateo, California: Morgan Kaufmann.

Doan, A., Domingos, P., & Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD Record*, *30*(2), 509–520.

Flesca, S., Manco, G., Masciari, E., Pontieri, L., & Pugliese, A. (2002). Detecting structural similarities between XML documents. In *Proceedings of the International Workshop on Web and Databases* (pp. 55–60). Madison, Wisconsin.

Fuhr, N., & Grossjohann, K. (2001). XIRQL: a query language for information retrieval in XML documents. In *Proceedings of the International Conference on Research and Development in Information Retrieval* (pp. 172–180). New York: ACM.

Fuhr, N., & Lalmas, M. (2004). Initiative for the evaluation of XML retrieval. http://inex.is.informatik.uni-duisburg.de:2004/.

Garofalakis, M. N., Gionis, A., Rastogi, R., Seshadri, S., & Shim, K. (2000). XTRACT: A system for extracting document type descriptors from XML documents. In *Proceedings of the International Conference on Management of Data* (pp. 165–176). New York: ACM.

Grahne, G., & Thomo, A. (2001). Approximate reasoning in semi-structured databases. In *Proceedings of the International Workshop on Knowledge Representation Meets Databases*, vol. 45 of *CEUR Workshop Proceedings*. Rome, Italy: CEUR-WS.org.

Guerrini, G., Mesiti, M., & Bertino, E. (2006). Structural similarity measures in sources of XML documents. In J. Darmont & O. Boussaid (Eds.), *Processing and Managing Complex Data for Decision Support* (pp. 247–279). Hershey, Pennsylvania: Idea Group.

Guerrini, G., Mesiti, M., & Sanz, I. (2006). An overview for clustering XML documents. In A. Vakali & G. Pallis (Eds.), *Web Data Management Practices: Emerging Techniques and Technologies* (pp. 56–78). Hershey, Pennsylvania: Idea Group.
Lee, M., Yang, L., Hsu, W., & Yang, X. (2002). XClust: Clustering XML schemas for effective integration. In *Proceedings of the International Conference on Information and Knowledge Management* (pp. 292–299). New York: ACM.
Lu, S. Y. (1979). A tree to tree distance and its applications to cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *1*, 219–224.
Madhavan, J., Bernstein, P., & Rahm, E. (2001). Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Databases* (pp. 49–58). San Francisco, California: Morgan Kaufmann.
Mesiti, M. (2002). A structural similarity measure for XML documents: theory and applications. Ph.D. dissertation, University of Genova, Italy. http://www.disi.unige.it.
Mignet, L., Barbosa, D., Veltri, P. (2003). The XML web: a first study. In *Proceedings of the International Conference on* WWW (pp. 500–510). New York: ACM.
Miller, A. (1995). WordNet: a lexical database for english. *Communications of the ACM, 38*(11), 39–41.
Moh, C., Lim, E., & Ng, W. (2000). Re-engineering structures from web documents. In *Proceedings of ACM DL* (pp. 67–76). New York: ACM.
Nestorov, S., Abiteboul, S., & Motwani, R. (1998). Extracting schema from semistructured data. In *Proceedings of the International Conference on Management of Data* (pp. 295–306). New York: ACM.
Nierman, A., & Jagadish, H. (2002). Evaluating structural similarity in XML documents. In *Proceedings of the International Workshop on Web and Databases* (pp. 61–66). Madison, Wisconsin.
Parent, C. & Spaccapietra, S. (1998). Issues and approaches of database integration. *Communications of the ACM*, *41*(5), 166–178.
Rahm, E., & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB J.*, *10*(4), 334–350.
Rice, S. V., Bunke, H., & Nartker, T. A. (1997). Classes of cost functions for string edit distance. *Algorithmica 18*(2), 271–280.
Schlieder, T. (2001). Similarity search in XML data using cost-based query transformations. In *Proceedings of the International Workshop on Web and Databases* (pp. 19–24). Santa Barbara, California.
Selkow, S. M. (1977). The tree-to-tree editing problem. *Information Processing Letters*, *6*(6), 184–186.
Stanoi, I., Mihaila, G., & Padmanabhan, S. (2003). A framework for the selective dissemination of XML documents based on inferred user profiles. In *Proceedings of the International Conference on Data Engineering* (pp. 531–542). Los Alamitos, California: IEEE Computer Society.
Tai, K.-C. (1979). The tree-to-tree correction problem. *Journal of the ACM*, *26*(3), 422–433.
Tanaka, E., & Tanaka, K. (1988). The tree-to-tree editing problem. *Journal of Pattern Recognition and Artificial Intelligence*, *2*(2).
Theobald, A., & Weikum, G. (2000). Adding relevance to XML. In *Proceedings of the International Workshop on the Web and Databases*, LNCS(1997) (pp. 105–124). Berlin Heidelberg New York: Springer.
Tversky, A. (1977). Features of similarity. *Journal of Psychological Review*, *84*(4), 327–352.
W3C. (1998). Extensible Markup Language (XML).
XML.org. (2003). XML.org focus areas. http://www.xml.org/xml/focus_areas.shtml.
Wang, K., & Liu, H. (1998). Discovering typical structures of documents: A road map approach. In *Proceedings of the ACM SIGIR* (pp. 146–154). New York: ACM.
Wang, Y., DeWitt, D. J., & Cai, J.-Y. (2003). X-Diff: an effective change detection algorithm for XML documents. In *Proceedings of the International Conference on Data Engineering* (pp. 574–580). Los Alamitos, California: IEEE Computer Society.
Yao, B., Ozsu, M., & Keenleyside, J. (2002). XBench–a family of benchmarks for XML DBMSs. In *Proceedings of EEXTT and DiWeb 2002*, vol. 2590 of *LNCS* (pp. 162–164). Berlin Heidelberg New York: Springer.
Zhang, K. (1993). A new editing based distance between unordered labeled trees. In *Proceedings of the Symposium on Combinatorial Pattern Matching*, vol. 684 of *LNCS* (pp. 110–121). Berlin Heidelberg New York: Springer.

Zhang, K., & Shasha, D. (1977). Tree pattern matching. *Pattern Matching Algorithms*. London, UK: Oxford University Press.
Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, *18*(6), 1245–1262.
Zhang, K., Shasha, D., & Wang, J.-L. (1994). Approximate tree matching in the presence of variable length don't cares. *Journal on Algorithms*, *16*(1), 33–66.
Zhang, K., Statman, R., & Shasha, D. (1992). On the editing distance between unordered labeled trees. *Information Processing Letters*, *42*(3), 133–139.