available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Computer Science Review

An overview on XML similarity: background, current trends and future directions

Joe Tekli, Richard Chbeir*, and Kokou Yetongnon

LE2I Laboratory UMR-CNRS, University of Bourgogne, 21078 Dijon Cedex France

ABSTRACT

In recent years, XML has been established as a major means for information management, and has been broadly utilized for complex data representation (e.g. multimedia objects). Owing to an unparalleled increasing use of the XML standard, developing efficient techniques for comparing XML-based documents becomes essential in the database and information retrieval communities. In this paper, we provide an overview of XML similarity/comparison by presenting existing research related to XML similarity. We also detail the possible applications of XML comparison processes in various fields, ranging over data warehousing, data integration, classification/clustering and XML querying, and discuss some required and emergent future research directions.

Keywords: XML, Semi-structured data; Structural similarity; Tree edit distance; Data warehousing; Document classification and clustering; Information retrieval; Ranked queries.

Contents

* Corresponding author. Tel.: +33 3 80 39 36 55; Fax: +33 3 80 39 68 69; e-mail: richard.chbeir@u-bourgogne.fr

1. Introduction

W3C's XML (eXtensible Mark-up Language) has recently gained unparalleled importance as a fundamental standard for efficient data management and exchange. Information destined to be broadcasted over the web is henceforth represented using XML, in order to guarantee its interoperability. The use of XML covers data representation and storage, database information interchange, data filtering, as well as web services interaction.

Owing to the increasing web exploitation of XML, XML-based similarity/comparison becomes a central issue in the database and information retrieval communities. By XML similarity, we underline XML document-related similarities, i.e., document/document, document/pattern[1], as well as document/grammar[2] comparison, the types of objects being compared varying w.r.t. (with respect to) the application scenario at hand. Applications of XML comparison are numerous and range over: i) version control, change management and data warehousing (finding, scoring and browsing changes between different versions of a document, support of temporal queries and index maintenance), ii) semi-structured data integration (identifying similar XML documents originating from different data sources, to be integrated so that the user can access more complete information), iii) classification/clustering of XML documents gathered from the web against a set of XML grammars declared in an XML database (just as schemas are necessary in traditional DBMS for the provision of efficient storage, retrieval and indexing facilities, the same is true for XML repositories), iv) as well as XML retrieval (finding and ranking results according to their similarity in order to retrieve the best results possible).

A wide range of algorithms for comparing semi-structured data, e.g., XML-based documents, have been proposed in the literature. These vary w.r.t. the kinds of XML data they consider, as well as the kinds of applications they perform. They can be classified in three main groups: i) Edit Distance (*ED*) based methods, ii) Information Retrieval (*IR*) based methods, and iii) other diverse approaches exploiting different techniques to XML comparison (e.g., edge matching, path similarity…). *ED*-based methods make use of dynamic programming techniques for finding the edit distance between tree structures, XML documents being modeled as ordered labeled trees. Most of these methods are designed for document/document comparison tasks. They target rigorously structured XML and are usually fine-grained. They are mainly useful for applications that require accurate (fine-grained) detection of XML document structural similarities, i.e., version control and change management (*ED* algorithms having the advantage of producing *edit scripts*, along the similarity value itself, which would be exploited in describing changes), data integration, as well as XML classification/clustering applications. *IR*-based approaches extend conventional information retrieval methods, e.g., the vector space model, so as to provide XML document/query similarity assessment. In this context, an XML query basically comes down to either an XML document, a pattern, or a conjunction of patterns (cf. Section 3.2). *IR*-methods target loosely structured XML data and are usually coarse-grained, thus useful and generally exploited for fast simple XML search and retrieval. Note that ranked XML querying applications usually prioritize performance w.r.t. result quality, i.e., producing *good-enough* results in short time laps. The third group of methods comprises of different approaches to XML similarity. They exploit various techniques (e.g., tag similarity, edge matching, path similarity, entropy…) addressing specific application scenarios. Some provide approximations of (more complex and accurate) existing approaches (mainly *ED*-based).

The goal of this study is to provide a unified view of the problem, assessing the different aspects, techniques and various applications related to XML similarity. The remainder of this paper is organized as follows. Section 2 presents a glimpse on XML as a data representation model. Section 3 reviews background in XML similarity, covering the three main groups of methods mentioned above: *ED*-based, *IR*-based and various application specific approaches. Section 4 develops the main applications and uses of XML comparison. Some ongoing motivations and possible future research directions are covered in the Section 5. Section 6 concludes the paper.

2. A Glimpse on XML

With the growth of the World Wide Web, there is an increasing need to automatically process Web documents for efficient data management, similarity clustering and search applications. While HTML (Hyper Text Markup Language) provides a rather visual markup, having knowledge of the logical structure of the data is a fundamental prerequisite for the interoperability of web-based information systems [31]. Hence, XML was introduced by the W3C as an efficient means for data representation and management.

2.1. Document-centric Vs Data-centric XML

There are two different views of XML: the *document-centric* view and the *data-centric* view.

- *Document-centric* design involves a liberal use of free-form text that is *marked up* with elements (cf. Figure 1.a). It focuses on XML applications for exchanging documents in the traditional sense. *Document-centric* XML is easy to render on some

[1] An XML pattern is a portion of an XML document.
[2] Either a DTD or XML Schema [28].

sort of output device. For example, it is quite easy to format the document in Figure 1.a. into HTML, because it is similar to HTML in its tagging and mark up styles.

– *Data-centric* XML, however, underlines well structured information, i.e. the data is strictly tagged (cf. Figure 1.b). The *data-centric* view is usually utilized for exchanging data in a structured form, such as classical EDI (Electronic Data Interchange) and database applications. In addition, *data-centric* documents are easier to process with computer programs and automatic processes, because the data is better organized.

```
<?XML>
  <Memo>
      Please make sure you are at the
      <Location>ninth floor</Location>by
      <MeetingTime>10:30 AM</MeetingTime>
      to<Purpose>discuss the
      budget</Purpose>
  </Memo>
```

a. *Document-centric* XML

```
<?XML>
  <Memo>
      <MeetingTime>10:30AM</MeetingTime>
      <Purpose>Discuss Budget</Purpose>
      <Location>ninth floor</Location>
  </Memo>
```

b. *Data-centric* XML

Fig. 1 - Sample *document-centric* and *data-centric* XML documents.

Both *data-centric* and *document-centric* aspects of XML have been considered in assessing XML similarity. As mentioned previously, *ED*-based XML similarity approaches are usually fine-grained and thus dedicated to comparing well structured *data-centric* XML documents, while *IR*-based methods are coarse-grained and target loosely structured *document-centric* XML.

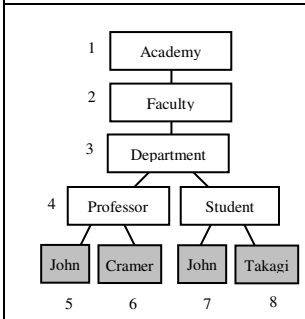
2.2. XML data model

XML documents represent hierarchically structured information and are generally modeled as Ordered Labeled Trees (OLTs)[1]. In a traditional DOM (Document Object Model) ordered labeled tree [88], nodes represent XML elements and are labeled with corresponding element tag names. Element attributes mark the nodes of their containing elements. Some studies have considered OLTs with distinct attribute nodes, labeled with corresponding attribute names [61], [90]. Attribute nodes appear as children of their encompassing element nodes, sorted by attribute name, and appearing before all sub-element siblings [61].

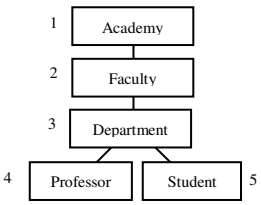
Element/attribute values can be disregarded (*structure-only*) or considered (*structure-and-content*) in the comparison process following the application scenario at hand (cf. Figure 2). In general, element/attribute values are disregarded when evaluating the structural properties of *heterogeneous* XML documents, i.e., documents originating from different data-sources and not conforming to the same grammar (DTD/XML Schema)[2], so as to perform XML structural classification/clustering [23] [61] or structural querying (i.e., querying the structure of documents, disregarding content [6]). Nonetheless, values are usually taken into account with methods dedicated to XML change management [16] [20], data integration [37] [51], and XML (*structure-and-content*) querying applications [74], [90], where documents tend to have relatively similar structures (probably conforming to the same grammar [47], [86]). With such methods, XML text sequences are usually decomposed into words, mapping each word to a leaf node labeled with the respective word.

```
<?XML>
  <Academy>
    <Faculty>
      <Department>
        <Professor>John Cramer </Professor>
        <Student>John Takagi </Student>
      </Department>
    </Faculty>
  </Academy>
```

a. XML document



b. OLT with values



c. OLTs where values are disregarded

Fig. 2. A sample XML document with corresponding OLTs.

3. Background on XML Similarity

Various criterions could allow the description and categorization of XML similarity methods, including:

i. The kind of technique being used: *ED*-based, *IR*-based, and others (tag similarity, edge matching, …)

ii. The kind of XML data being compared:
   – document/document, document/pattern or document/grammar
   – document-centric or data-centric,
   – structure-only or structure-and-content

iii. The intended application domain: change management and version control, data integration, classification/clustering, and ranked querying.

[1] In the following, *tree* designates *ordered labeled tree*.

[2] It is the case of lots of XML documents on the web [61].

In the following, for clarity of presentation, we review XML similarity methods based on the kinds of techniques they exploit (i.e., *ED*-based, *IR*-based and others). We estimate this categorization provides the simplest and most consistent unified view of the wide variety of divers methods proposed in the literature. The kinds of XML data being treated as well as the intended applications domains will be discussed for each method. Catalogs summarizing the properties and characteristics of all methods covered in this review are depicted in Tables 1, 2 and 3 at the end of the paper.

3.1. Tree edit distance methods for XML similarity

Various methods, for estimating the similarities between hierarchically structured data, particularly between XML documents, have been proposed in the literature. Most of them derive, in one way or another, the dynamic programming techniques for finding the edit distance between strings [48], [83], [87]. In essence, all these approaches aim at finding the cheapest sequence of edit operations that can transform one tree into another, XML documents being modelled as ordered labelled trees [88]. Below, we provide formal definitions of the common concepts related to tree *ED*.

3.1.1. Basic notions and common concepts

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

Definition 2 - Edit distance: The edit distance between two trees A and B is defined as the minimum cost of all edit scripts that transforms A to B: $Dist(A, B) = Min\{Cost_{ES}\}$. Thus, the problem of comparing two trees A and B, i.e. evaluating the structural similarity between A and B, is defined as the problem of computing the corresponding tree edit distance [89].

As for tree edit operations, they differ following the edit distance method at hand, and can be classified in two groups: *atomic* tree edit operations and *complex* edit operations. An atomic edit operation on a tree (i.e. rooted ordered labeled tree) is either the deletion of an inner/leaf node, the insertion of an inner/leaf node, or the replacement (i.e. update) of a node by another one. A complex tree edit operation is a set of atomic tree edit operations, treated as one single operation. A complex tree edit operation is either the insertion of a whole tree as a sub-tree in another tree (which is actually a sequence of atomic node insertion operations), the deletion of a whole tree (which consists of a sequence of atomic node deletion operations), or moving a sub-tree for one position into another in its containing tree (which is a sequence of atomic node insertion/deletion operations). The authors in [17] introduce sub-tree copying and gluing operations. These are similar to tree insertions/deletions respectively, but are defined in the context of unordered tree comparison. Thus, they won't be further investigated hereunder.

In the following, we present the general form and variations for each of the tree edit operations stated above.

Definition 3 – Update node: Given a node x in tree T and a new node y, *Upd(x, y)* is an update operation replacing x by y in the transformed tree (cf. Figure 3.a). Node y will have the same parent and children as x.

Definition 4 – Insert node:
Variation 1: Given a node x and a tree T, T encompassing node p with first level sub-trees (i.e. children) P_1, …, P_m, *Ins(x, p, {P_i, …, P_j})* inserts node x in T as the i^{th} child of p making x the parent of the consecutive subsequence of sub-trees $\{P_i, …, P_j\}$ of p (cf., Figure 3.b).
Variation 2: The insertion operation can be restricted to leaf nodes in some *ED* approaches: *InsLeaf(x, p, i)* inserting leaf node x as the i^{th} child of p (cf., Figure 3.d)

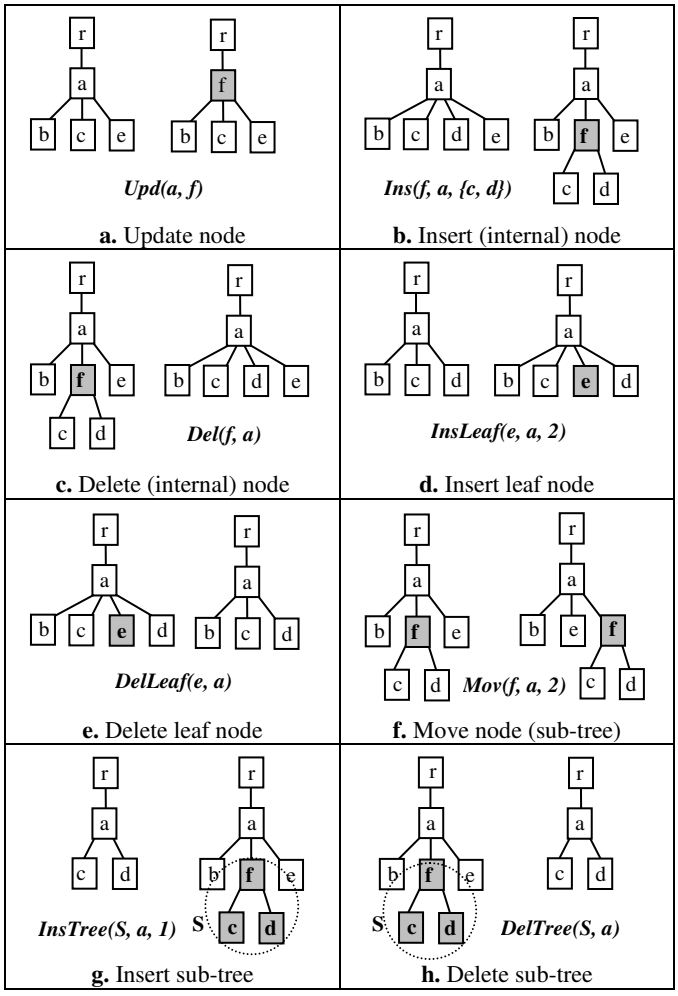


Fig. 3 - Tree edit operations (simplified syntaxes).

Definition 5 – Delete node:
Variation 1: Given a node x and a tree T, T encompassing node p with first level sub-trees (i.e. children) $\{P_1, \ldots, P_{i-1}, x, P_{i+1}, \ldots, P_m\}$, and x having first level sub-trees $\{X_1, \ldots, X_n\}$, *Del(x, p)* deletes node x in T, making the children of x become the children of p. The children of x are inserted in the place of x as a subsequence in the left-to-right order of the children of p (Figure 3.c).
Variation 2: The deletion operation can be restricted to leaf nodes in some *ED* methods: *DelLeaf(x, p)* (cf., Figure 3.e).

Definition 6 - Insert tree: Given a tree A and a tree T, T including node p with first level sub-trees $\{P_1, \ldots, P_m\}$, *InsTree(A, p, i)* is a tree insertion applied to T, inserting A as the i^{th} sub-tree of p (cf. Figure 3.g).

Definition 7 - Delete tree: Given a tree A and a tree T, T encompassing node p with first level sub-trees $\{P_1, \ldots, P_{i-1}, A, P_{i+1}, \ldots, P_n\}$, *DelTree(A, p)* deletes sub-tree A in T from among the children of p (cf. Figure 3.h).

Definition 8 – Move node: Given a node x and a tree T, T encompassing node p, *Mov(x, p, i)* moves x and its children to become the i^{th} child of node p (cf. Figure 3.f).

3.1.2. State of the art in tree edit distance methods

Tree *ED* algorithms can be distinguished by the set of edit operations that are allowed as well as overall complexity/performance and optimality/efficiency level.

Early approaches: In [78], the author introduces the first non-exponential algorithm to compute the edit distance between ordered labeled trees, allowing insertion, deletion and substitution (relabeling) of inner nodes and leaf nodes. The resulting algorithm has a complexity of $O(|T_1|\times|T_2|\times depth(T_1)^2\times depth(T_2)^2)$ when finding the minimum edit distance between two trees T_1 and T_2 ($|T_1|$ and $|T_2|$ denote tree cardinalities while $depth(T_1)$ and $depth(T_2)$ are the depths of the trees). Similarly, early approaches in [77] and [89] allow insertion, deletion and relabeling of nodes anywhere in the tree (cf. Figures 3.a, 3.b and 3.c). Yet, they remain complex. The approach in [89] is of complexity $O(e^2\times|T_1|\times min(|T_1|, |T_2|))$ (e is the weighted edit distance[1]), while that in [77] has a time complexity of $O(|T_1|\times|T_2|\times depth(T_1)\times depth(T_2))$. Note that the approaches in [77], [78], [89] were not mainly developed in the XML context, and thus might yield results (i.e. edit scripts, and consequently distances) that are not completely appropriate to XML data. In particular, it has been recently argued that restricting insertion and deletion operations to leaf nodes fits better in the context of XML data, w.r.t. insertions and deletions applied anywhere in the XML tree [23]. Following [23], the latter pair of operations destroys the membership restrictions of the XML hierarchy and thus does not seem to be natural for XML data (e.g., deleting/inserting an inner node and moving its children up/down one level, cf. Figures 3.d and 3.c). *ED*-based approaches dedicated to XML-based data tend to prevent such operations by utilizing ones that target leaf nodes (cf. Figures 3.d and 3.e) or whole sub-trees (cf. Figures 3.g and 3.h). Hence, the deletion of an internal node in an XML document tree would require deletions of all nodes in its path, starting from the leaf node and going up to the internal node itself, which could also be performed via one single tree deletion operation. Likewise, the insertion of an inner node must be performed before the insertion of any of its descendents, which could be undertaken via one tree insertion operation.

Trading quality for performance: In [16], [20], the authors restrict insertion and deletion operations to leaf nodes and add a move operator that can relocate a sub-tree, as a single edit operation, from one parent to another (cf. Figures 3.a, 3.d, 3.e and 3.f). Note that a move operation can be seen as a succession of insert and delete operations. Yet, it is different in its cost (the authors in [16], [20] consider that the cost of moving a sub-tree is much lesser than the sum of the costs of deleting the same sub-tree, node by node, from its current position and inserting it in its new location). However, algorithms in [16], [20] do not guaranty optimal results.

On one hand, the algorithm in [16] runs in five phases: *update*, *align*, *insert*, *move* and *delete*. Each of them, to the exception of the *align* phase, corresponds to the application of the best possible combinations of the corresponding edit operation, so as to match the compared trees. As for the *align* phase, the authors make use of a variation of the Longest Common Subsequence algorithms (LCS) [59] in determining the set of misaligned children nodes[2] of a given inner node (repeated for each tree node throughout the matching process). Those misalignments are exploited in the subsequent *move* phase in determining the set of move operations to be performed. Nonetheless, XML documents being compared should abide two main assumptions without which the algorithm would yield suboptimal results (i.e., overlooking the minimum cost edit script): i) node labels have to follow a certain predefined ordering w.r.t. a given schema, and ii) given any leaf node in the first document, there is at most one leaf node in the second document that matches the first and vice versa. Algorithm complexity simplifies to $O(n\times e + e^2)$, where n is the total number of leaf nodes in the trees being compared.

On the other hand, the algorithm in [20] tries to detect large sub-trees that were left unchanged between the two XML trees being compared. These are matched. Consequently, it tries to match more nodes by considering ancestors and descendants of matched nodes, taking labels

[1] Let $S = op_1, op_2, \ldots, op_n$ be the cheapest sequence of edit operations that transforms tree A to B, then the weighted edit distance is given by $e = \sum_{1\le i\le n} w_i$ where w_i, for $1\le i \le n$, is 1 if op_i is an insert or delete operation, and 0 otherwise.

[2] These are nodes bearing the same labels/values but having different ordering positions w.r.t. their parent nodes.

into account. It also considers ID attributes in matching corresponding nodes (nodes with identical IDs are matched). The algorithm however trades some quality to get an algorithm which runs in average linear time: no more than $O(N \times log(N))$ where N is the maximum number of nodes in the trees being compared. In other words, the distance attained when comparing two trees is not always minimal, some sets of move operations not being optimal (i.e., the produced edit script is not of minimal cost).

Both methods in [16] and [20] were developed in the context of change management and version control. They are designed for document/document comparison and consider XML element/attribute values (cf. Figure 2.b) in their computations, in contrast with remaining methods in this section which specifically target the structural properties of XML documents (cf. Figure 2.c).

Combining efficiency and performance: Work provided in [18] has been considered as a reference point in recent tree edit distance literature and has provided the basis for various XML related structural comparison studies [61], [23], [79]. Chawathe's approach restricts insertion and deletion operations to leaf nodes (which are viewed as natural operations in the XML context) and allows the relabeling of nodes anywhere in the tree, while disregarding the move operation (cf. Figures 3.a, 3.d and 3.e). The proposed algorithm is a direct application of the famous Wagner-Fisher algorithm [83] which optimality has been accredited in a broad variety of computational applications [2], [87]. It is also among the fastest tree *ED* algorithms available. In short, the author transforms trees into special sequences called ld-pairs. The ld-pair representation of a tree comes down to the list, in preorder, of the ld-pair representations of its nodes. The ld-pair representation of a tree node is the pair *(l, d)* where *l* is the node's label and *d* its depth in the tree (e.g., the ld-pair representation of the XML tree in Figure 2.c is *(Academy, 0), (Faculty, 1), (Department, 2), (Professor, 3), (Student, 3)*). Consequently, the author simplifies the problem of comparing two document trees to that of comparing the corresponding ld-pair representations, using a specialization of the Wagner-Fisher algorithm [83]. He also extends his algorithm for external-memory computations and identifies respective I/O, RAM and CPU costs. Note that this is the only algorithm that has been extended to efficiently calculate edit distances in external memory (without any loss of computation quality/efficiency).
The overall complexity of Chawathe's algorithm is of $O(N^2)$, its performance and efficiency being recognized in [61] as well as [23] (recall that N is the maximum number of nodes in the trees being compared).

Sub-tree similarity: While it might be considered as a starting point for recent XML tree edit distance approaches, the approach in [18] overlooks certain sub-tree similarities while comparing XML documents. For instance, computing edit distance between XML trees *A*, *B* and *C* in Figure 4 yields $Dist(A, B) = Dist(A, C) = 3$, which corresponds to the cost of three consecutive insert operations (unit costs are usually used) introducing nodes *b*, *c* and *d* (*e*, *f* and *g*) in tree *A* transforming it into *B* (*C*). Nonetheless, one can realize that tree *A* is structurally more similar to *B*, than to *C*, the sub-tree A_1, made up of nodes *b*, *c* and *d*, appearing twice in *B* (B_1 and B_2) and only once in *C* (C_1). Such sub-tree structural similarities are also left unaddressed by other existing approaches.

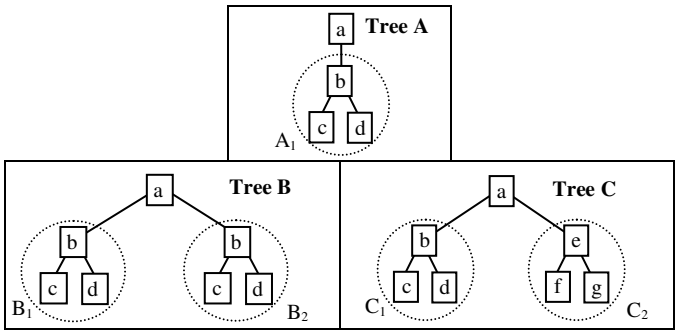


Fig. 4 - Sample XML trees.

In [61], the authors stress the importance of identifying sub-tree structural similarities in an XML tree comparison context, due to the frequent presence of *repeated* and *optional* elements in XML documents. Repeating elements often induce multiple occurrences of similar element/attribute sub-trees (presence of optional elements/attributes) or identical sub-trees in the same XML document (such as the sub-trees B_1 and B_2 in XML tree *B*, cf. Figure 4) which reflects the need to take these sub-tree resemblances into consideration while comparing documents. The authors in [61] extend the approach of Chawathe [18] by adding two new operations: insert tree and delete tree (cf. Figures 3.g and 3.h) to discover sub-tree similarities, by making use of the *contained in* relation between trees/sub-trees. A tree T_1 is said to be *contained in* a tree T_2 if all nodes of T_1 occur in T_2, with the same parent/child edge relationship and node order. Additional nodes may occur in T_2 between nodes in the embedding of T_1.
Following [61], a tree *A* may be inserted in *T* only if *A* is already *contained in* the source tree *T*. Similarly, a tree *A* may be deleted only if *A* is already *contained in* the destination tree *T*. Therefore, the proposed approach captures the sub-tree structural similarities between XML trees *A/B* in Figure 5, transforming *A* to *B* in a single edit operation: (inserting sub-tree B_2 in *A*, B_2 occurring in *A* as A_1), whereas transforming *A* to *C* would always need three consecutive insert operations (inserting nodes *e*, *f* and *g*).
The overall complexity of their algorithm simplifies to $O(N^2)$. Structural clustering experiments in [61] show that the proposed algorithm outperforms, in quality, that of Chawathe [18], which in turn yields better results than Zhang and Shasha's algorithm [89]. However, the authors in [61] show that their algorithm is conceptually more complex than its predecessor, requiring a pre-computation phase for determining the costs of tree insert and delete operations (which complexity is of $O(2 \times N + N^2)$).

Structural summaries: The authors in [23] provide an edit distance algorithm combining features from both [18], [61] and propose to apply it on XML tree structural summaries, instead of whole document trees, in order to gain in performance. Structural summaries are produced using a dedicated repetition/nesting reduction process. The structural summary of an XML tree comes down to a modified tree in which the redundancies due to nested-repeated and repeated XML nodes are eliminated (e.g., the structural summary of tree *B* in Figure 4 is tree *A*, the repeated sub-tree B_2 being omitted). Their algorithm can be viewed as a special case of [61]'s algorithm where insert and delete tree operations costs are computed as the sum of the costs of inserting/deleting all individual nodes in the considered sub-trees. The algorithm is of $O(N^2)$ complexity. Experimental results in [23] showed improved document clustering quality w.r.t. [18]'s algorithm.

XML/DTD similarity: In [80], the authors propose an *ED*-based approach that is slightly different in scope w.r.t. existing methods. The authors provide an algorithm dedicated to comparing an XML document and a DTD grammar. They introduce an algorithm based on the tree edit distance concept, as an effective and efficient means for comparing tree structures, XML documents and DTDs being modeled as ordered labeled trees (element/attribute values in elements are disregarded). The proposed method extends the algorithm in [61] by considering the various DTD constraints on the existence, repeatability and alternativeness of XML elements/attributes. The approach is of polynomial complexity ($O(N^3)$ where N is the maximum number of nodes in the XML/DTD trees being compared), in comparison with existing exponential methods, i.e., [6], presented in the following section. Classification experiments on sets of real and synthetic XML documents, underline the approach's effectiveness, and its applicability to large XML repositories.

Comparing tree edit distance approaches, to identify the best XML structural similarity methods, is not a trivial task. Each method is developed in a specific context and might thus yield inappropriate results when applied in a different framework. Nonetheless, w.r.t. the two major criterions that roughly characterize tree edit distance methods: *efficiency* (quality) and *performance* (complexity), the approach in [61] seems to be one of the most sophisticated *ED*-based XML structural similarity methods to date. Regarding quality, Nierman and Jagadish's approach [61] was proven to be more accurate in detecting XML structural resemblances, particularly sub-tree related similarities, in comparison with some of its famous predecessors (i.e. [18], [89]). It was consequently adopted as the basis for more recent studies, particularly [23], [80]. Regarding performance, the authors in [61] were able to maintain quadratic time complexity in developing their algorithm, which is typical to *ED*-based approaches.

Table 1 depicts the various *ED*-based XML similarity approaches developed in the literature.

3.2. Information retrieval methods for XML similarity

While a lot of work has been undertaken in edit distance related research, for comparing XML data, XML similarity is also becoming one of the central topics in the information retrieval field. Recall that *ED*-based approaches focus on rigorously structured *data-centric* XML and target change management, data integration and structural classification/clustering applications, whereas *IR*-based methods treat loosely structured *document-centric* XML and mainly target ranked XML querying.

Since documents are flat with conventional information retrieval, i.e. they represent unstructured data, traditional *IR* methods for searching and querying information are no longer adequate with semi-structured data such as XML [36]. As stated earlier, XML documents represent hierarchically structured information (cf. Figure 2). In other words, content is distributed at different levels of the document tree. Therefore, it is to be treated differently w.r.t. flat content so as to improve retrieval precision [75]. First, information placed near the root node of an XML document tends to be more important than information further down in the hierarchy [6] [90]. Intuitively, as one descends in the XML tree hierarchy, information becomes increasingly specific, consisting of finer and finer details, its affect on the meaning of the whole document tree decreasing accordingly. This is orthogonal to traditional flat documents where information is placed at one single level and is of the same relevance. Second, users in turn would like to refer to document structure when searching for relevant information in XML documents [36]. To do so, they pose so called *content-and-structure* queries, in comparison with the *content-only* queries in conventional *IR* [69], by restricting the context of interest to some XML elements in the documents being searched [36]. Therefore, various attempts to extend existing *IR* methods in order to account for the structure of XML documents, in the document/query comparison process, have been undertaken (a query, in the *IR* context, underlining a whole XML document, an XML document fragment or a conjunction of fragments).

In the following subsection, we start by presenting an overall view of traditional *IR* concepts, the vector space model in particular. Thereafter, we cover *IR* methods extended for XML data.

3.2.1. Traditional information retrieval

Information retrieval (*IR*) is a branch of informatics concerned with the acquisition, organization, storage, search and selection of information [69]. While it is used to deal with flat textual data (i.e. classical free text documents), *IR* is being extended, since the last two decades, so as to treat complex information such as structured/semi-structured data (e.g. XML, SGML, HTML, etc.), images, graphics, sounds and videos.

In essence, the goal of *IR* is to efficiently identify/retrieve, in a data collection, information that is

relevant w.r.t. the user's needs [7]. While relevance in *IR* is a broad and imprecise notion, the abstract concept of *relevance* is generally concretized by the notion of *similarity* [65]. With conventional *IR*, documents and user queries usually consist of sets of keywords. Thus, identifying documents that are relevant (similar) to a given query comes down to:

- Comparing the keywords of each document in the document collection to those of the query,
- Ranking the documents w.r.t. their keyword similarities with the query (document selection is undertaken by defining a similarity threshold, e.g. range queries [1] or KNN queries [68]).

Keywords are commonly weighted in order to reflect their relative importance in the query/document at hand. The underlying idea is that terms that are of more importance in describing a given query/document are assigned a higher weight. As a weighting scheme, the standard *TF-IDF* (*Term Frequency – Inverse Document Frequency*) approach (and its variants) of the *vector space model* [69] [70] is usually used.

Note that various *IR* models, other than the vector space model, have been proposed in the literature, among which the Boolean model [46], the probabilistic model [30], the LSI (Latent Semantic Indexing) model [24], the DFR (Divergence From Randomness) model [3], etc. However, in this chapter, we restrict ourselves to the vector space model since it is the most commonly used, its performance being accredited in a broad variety of applications and scenarios (e.g., [19], [21], [27]).

Most attempts to extend *IR* techniques, so as to account for XML data, focus on *TF-IDF* and the *vector space model*. Therefore, we briefly review those notions prior to discussing XML *IR* approaches. In the standard vector space model, documents and queries are indexed in a similar manner, producing vectors in a space which dimensions represent, each, a distinct indexing unit t_i. An indexing unit usually stands for a single term, i.e. a keyword[1]. The coordinate of a given document D on dimension t_i, is noted $w_D(t_i)$ and stands for the weight of t_i in document D within a document collection. $w_D(t_i)$ is computed using a score of the *TF-IDF* family, taking into consideration both document and collection statistics. Consequently, the relevance of a document D to a query Q, designated as $Sim(Q, D)$, is evaluated using a measure of similarity between vectors such as the inner product, the cosine measure, the Jaccard measure, the Dice coefficient, etc., [7], [32], [52]. For instance, the cosine measure, which is one of the most commonly used in the IR literature, is expressed as follows:

$$\mathrm{Cos}(\vec{Q},\vec{D}) = \frac{\sum_{r=1}^{M} w_Q(n_r) \times w_D(n_r)}{\sqrt{\sum_{r=1}^{M} w_Q(n_r)^2 \times \sum_{r=1}^{n} w_D(n_r)^2}} \in [0,1] \qquad \textbf{(1)}$$

[1] A keyword can also consist of multiple words (phrase units).

As for the *TF-IDF* score, different variations have been proposed [69], [70], [71]. We give below the standard definition. The *TF-IDF* score, making up weight $w_D(t_i)$, comprises of two factors [69]:

- The *TF* (*Term Frequency*) factor which designates the number of times a term t_i occurs in document D (document statistics). The underlying idea is that the importance of a given term t_i in describing a document D increases with the frequent use of t_i in D.
- The *IDF* (*Inverse Document Frequency*) factor, emphasizing the fraction of documents that contain term t_i (collection statistics). The underlying idea is that the importance of a given term t_i in describing a document D decreases with the frequent use of t_i in the document collection.

A common *TF-IDF* mathematical formulation [70], [71] would be as follows. $w_D(t_i) = TF \times IDF$ having:

- $TF = tf(t_i, D)$ underlining the number of times term t_i occurs in D
- IDF = $\log \frac{N}{df(t_i, D)}$ where N is the total number of documents in the document collection, and $df(t_i, D)$ is the number of documents containing term t_i

The *TF-IDF* score could also be normalized as detailed in [69], for example:

$$w_D(t_i) = \frac{tf(t_i, D) \times \log \frac{N}{df(t_i)}}{\sqrt{\sum_{t_i} \left(tf(t_i, D) \times \log \frac{N}{df(t_i)} \right)^2}} \qquad \textbf{(2)}$$

Because of its well known efficiency when dealing with flat textual data, various studies have focused on extending the vector retrieval *TF-IDF* based approaches so as to treat semi-structured data, i.e. XML. We cover those main approaches in the following section.

3.2.2. Extending conventional information retrieval to deal with XML

A number of techniques extending the vector space model towards effective XML information retrieval have been designed, namely [13], [31], [36], [75], [62]. Note that various other techniques, e.g. [4], [56] …, have also been proposed. Nonetheless, we limit our presentation here to the most basic studies, remaining approaches being covered in the applications' section.

Indexing nodes: In [31], Fuhr and Großjohann define so-called *indexing nodes*. These are atomic units in the XML document, basically XML elements, which encompass disjoint sub-trees. Given these nodes, *TF-IDF* weights can be computed locally, instead of being

evaluated w.r.t. to the whole document. The weights would then be *augmented*, down-weighting the statistics (*TF-IDF* values) when the terms are propagated upwards in the document tree, thus taking into account the hierarchical aspect of XML. The underlying idea is that: the larger the distance between a node and its ancestor, the lesser that node should contribute to the relevance of its ancestor's content [31]. Following Fuhr and Großjohann, the *indexing nodes* are to be identified explicitly in a dedicated XML grammar definition (that the authors identify as an *extended DTD*) corresponding to the XML document at hand (for instance, the following hypothetical element declaration *<!ELEMENT Professor (#INDEX)>* could designate that *Professor* elements in the XML document of Figure 5 are indexing nodes).

For example, in Figure 5, nodes *Professor* and *Student* are considered as *indexing nodes*. Thus, *TF-IDF* scores are computed for these nodes based on their textual content (e.g. for node *Professor*, scores for *"John"* and *"Cramer"* are computed). Then, these scores are propagated to upper nodes in the document tree, multiplying them by corresponding *augmentation weights* (e.g. for node *Professor*, *TF-IDF* scores are roughly multiplied by a factor of *0.6* when propagated to node *Department*, thus decreasing their relevance w.r.t. *Department*).

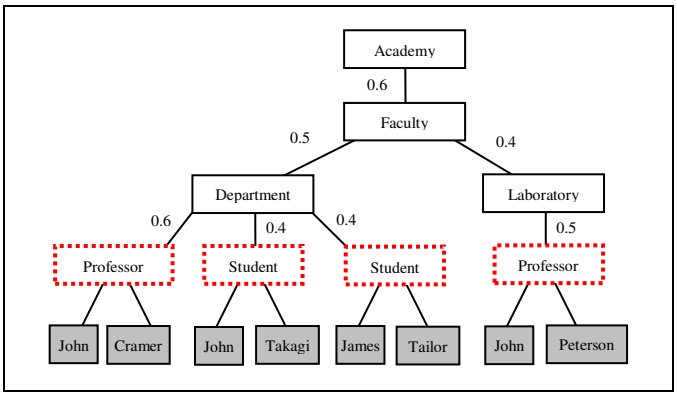


Fig. 5 - XML document tree with predefined *augmentation weights*.

Note that while it employs *TF-IDF* scoring, the approach in [31] is also built on concepts of the probabilistic *IR* model, using facts and rules to map the XML content and structure. Nonetheless, it constitutes the building blocks for a well known vector retrieval technique, i.e. [36].

Single/multi-category retrieval: In [36], Grabs and Schek build on Fuhr and Großjohann's approach [31] by introducing the notion of *category*. Following Grabs and Schek, users may want to refer to a given part (category) of the XML document in isolation, requesting only content that is relevant to that category, i.e. *single category retrieval* (e.g. in Figure 5, the user might be interested in professors that only work in laboratories. Thus, only the contents of nodes *Professor* corresponding to category *Laboratory* should be considered when computing term statistics. In other words, the score for term "John" – corresponding to *Professor John Peterson* – should not be affected by occurrences of "John" in elements outside the category *Laboratory*). Likewise, in other cases, users may request information from several categories, or they do not care to which category the requested content belongs, i.e. *multi-category retrieval* (e.g. in Figure 5, the user might be interested in all professors. Thus term statistics should be assessed taking into account all corresponding categories). To solve the category problem, the authors propose to keep the indexes for basic nodes (*indexing nodes* [31]) and to derive required indexes and statistics from the underlying basic ones on-the-fly, i.e. at query runtime. In other words, the vector space is generated dynamically following user information requests. That is what the authors identify as *flexible retrieval*, i.e. the users can dynamically – at query time – define the scope of their queries.

Indexing based on term context: In an attempt to further account for the XML structure in *IR*, Carmel *et al.* [13] propose to extend the vector space model by replacing the basic indexing units, i.e. terms t_i, by pairs of the form (t_i, c_i) where each term t_i is qualified by the context in which it appears. The context of appearance of a term is the path for navigating the hierarchical structure of the XML document, from the root node to the node in which term t_i occurs (e.g. in Figure 5, the first occurrence of term *John* will be associated with the path */Academy/Faculty/Department/Professor* as its context). The authors propose to compute weights of the form $w_D(t_i, c_i)$ and to extend query/document vectors accordingly. In addition, they suggest relaxing the query/document cosine similarity measure (more precisely the scalar product part) by accounting not only for exact "term in context" matching but also for context resemblance. They make use of a dynamic programming LCS (Longest Common Subsequence) algorithm [41] to compute similarity values between contexts (paths), which are subsequently integrated in the cosine measure.

Structural term indexing: Another approach, extending the vector space model to incorporate XML document structure, is provided by Schlieder and Meuss in [75]. The authors extend the standard notion of term t_i to *structural term* T_i, a structural term being a labeled tree. Note that in [75], queries are represented as labeled trees (thus including structural terms) and the query model is based on tree matching (the *unordered tree inclusion* variant is adopted [57]) as a simple means for formulating queries without knowing the exact structure of the XML data. Subsequently, the authors adapt the notions of term frequency (*TF*) and inverse document frequency (*IDF*) to the structural terms: a structural term *T* occurs in an XML document *D* if it matches D following the *unordered tree inclusion* matching operator (e.g., structural term *Professor/John* occurs two times in the XML document in Figure 5. Its *TF* score w.r.t. to the document at hand is equal to *2*. IDF scores are computed in the same manner).

Flexible retrieval: In addition, as in [36], Schlieder and Meuss enable *flexible querying* in [75] by introducing the notion of *logical XML document*. A *logical XML document* in a document collection is none other than a sub-tree of that collection (e.g. in Figure 5, not only the sub-tree rooted at the *Academy* node is a logical document, but also all its sub-trees, that is the sub-trees rooted at *Faculty*, *Department* and *Laboratory*, as well as each of the *Professor* and *Student* nodes). As a result, the user defines, via her query, the kind of logical documents to be retrieved. These are the documents having roots identical to that of the query. Similarly to [36], *TF-IDF* scores are all computed at query time. The authors in [75] also demonstrate that, by adjusting parameters of the retrieval process, their method can model both the classical vector space model (i.e. use of classical terms t_i and *TF-IDF* statistics) and the original tree matching approach (*unordered tree inclusion*).

Matrix model: In [62], Pokorny and Rejlek consider that previous XML *IR* methods, such as [13], [31], [36], do not sufficiently consider the structure of XML documents in the retrieval process. In their study, Pokorny and Rejlek represent XML documents as matrices instead of simple vectors. In the vector space model, a weight $w_D(t_i)$ is expressed by a real number (e.g. *TF-IDF* score) specifying the distribution of a term t_i for the entire document. In the matrix model [62], the weight of a term is expressed by a vector $w_D(t_i)_{1...k}$. Such a weight should reflect the distribution of the term t_i in the XML structure of the document collection, w.r.t. to each path k occurring in the collection. Note that for simple path representations, the authors rely on a known DataGuide technique [34], since the main idea of a DataGuide is to provide a summary of the structure of a document collection. As documents and queries are represented as matrixes, the similarity between a query and a document is evaluated as the correlation between corresponding matrixes, providing the basis for an enhanced structure-aware query system [62].

Due to its novelty and relevance, XML *IR* is still very much in flux, new approaches being proposed regularly, which makes it obviously difficult to compare the various methods. The criterions needed to conduct the comparison, as well as the corresponding experimental framework, are continuously debated, namely in the INEX[1] campaigns dedicated to XML retrieval. Consequently, while it might seem early to survey XML *IR* at this point in time, we feel that a simple presentation of the proposed studies and their applications would motivate further innovations in the field.

Table 2 covers the central *IR*-based XML similarity methods developed in the literature.

[1] INitiative for the Evaluation of XML Retrieval, http://inex.is.informatik.uni-duisburg.de/.

3.3. Other techniques for XML similarity

While *ED* and *IR*-based XML comparisons cover a wide array of studies in the literature, several other approaches for evaluating XML similarity have been developed. Each of those exploits a different kind of technique (e.g., tag similarity, edge matching, path similarity…) and is dedicated to a specific application, some providing approximations of more complex existing approaches (mainly *ED*-based).

Note that due to the diversity of their underlying techniques, it is not a trivial task to categorize the various methods presented in this section. A classification based on application area is both difficult and restrictive since the same method could be exploited in various application domains (despite being developed or tested in a specific context). Thus, for clarity of presentation, we categorize methods following the nature of the XML data they treat: *structure-only* (disregarding XML element/attribute values) and *structure-and-content* similarity methods.

3.3.1. Structure-only XML similarity methods

XML element/attribute values are generally disregarded when evaluating the structural properties of heterogeneous XML documents, i.e., documents originating from different data sources and thus not conforming to the same grammar (DTD and/or XML Schema). Such methods are generally suitable for structural classification/clustering and XML structural querying applications (cf. Section 2.2).

Tag similarity: Since optimal *ED* algorithms usually require $O(N^2)$ [10] (complexity with early algorithms reaching $O(N^4)$, as shown in Section 3.1.2), various alternatives and approximations of the *ED* computational techniques have been developed in the literature, so as to reduce complexity. In particular, tag similarity is considered as the simplest measure for XML similarity, as it only evaluates how closely the set of element/attribute tags match between two XML documents. It was proposed as an alternative to more complex structural similarity methods, particularly *ED*-based, in the context of XML document clustering [10], [61]. In short, it considers the intersection of the sets of tags, between the documents being compared, over the union. Nonetheless, using tag similarity, the structure of the documents is completely ignored, thus attaining low clustering quality (i.e., generated clusters do not correspond to the predefined ones) in comparison with *ED*-based methods [10], [61].

Edge matching: In [45], Kriegel and Schönauer combine the simple node (tag) matching technique (estimating similarity between two XML documents based on their matching nodes, w.r.t. to specific node matching criterions – basically tag equality, completely ignoring the structure of the documents) and the *ED* concept. The authors put forward the *edge matching* approach: matching

the edges connecting XML nodes, thus taking into account the XML document's structure in the comparison process. The authors in [45] demonstrate that the edge matching approach is a lower bound of the *ED* techniques (less accurate), and that it is of $O(E^3)$ complexity (E is the maximum number of edges in the documents being compared, having $E=N-1$ for XML trees, where N is the maximum number of nodes). A similar edge matching approach is provided in [49] where authors represent XML documents as directed graphs (i.e graphs with directed edges) and define a distance metric that captures the number of common edges between the graph representations of two XML documents.

$$Dist(G_1, G_2) = \frac{1 - Edges(G_1) \cap Edges(G_2)}{Max\{Edges(G_1), Edges(G_2)\}} \quad \textbf{(3)}$$

The authors show that the proposed metric is efficient in clustering XML documents, with respect to *ED*-based methods. The authors demonstrate that their approach is of $O(N^2)$ complexity in the worst case scenario and state that it usually comes down to $O(k \times N)$ where k is a small multiple of N (recall that N is the maximum number of nodes in the XML documents being compared)

Path similarity: The authors in [10], [64] describe the structure of an XML document as a set of paths (starting from the root node and ending in the leaf nodes of the XML document tree, taking into account all the paths in between, e.g. the path set of the second XML tree in Figure 3.h is *{r/a/c, r/a/d, r/a, a/c, a/d}*). Subsequently, XML documents are compared w.r.t. their corresponding sets of paths: the more paths two XML documents share in common, the more similar they are. The path similarity method is shown to be of linear time complexity [64]. Its complexity can be reduced to $O(1)$ when coupled with the shingle technique [9] to create constant sized representations of arbitrary documents [10]. XML document clustering experiments in [10] and [64] show that the path similarity method provides fairly accurate similarity results w.r.t. tree *ED* comparisons.

In [42], the authors extend a variant of the *set of paths* technique and consider *sets of XPaths*[1]. Classic paths underline parent/child relationships in the XML document tree, ignoring sibling information. Nonetheless, *XPaths* (e.g., *r[1]/a[1]/c[1], r[1]/a[1]/d[1]*, describing XML document tree in Figure 3.h) incorporate some sibling information. An *XPath* underlines, for a given node, how many preceding siblings have the same label (It does not capture sibling information about nodes whose labels are different from the given node, which is explicitly stated by the authors). Experimental results in [42] show that the *set of XPaths* approach (describing XML documents as *sets of XPaths* and consequently comparing the corresponding sets) yields better clustering quality than the classic *set of paths* variant considered in the study.

[1] http://www.w3c.org/TR/xpath

Set similarity: In [11], Candillier *et al.* represent XML documents as sets of attribute-values, including: the set of parent-child relations (i.e., edges), the set of next-sibling relations and the set of distinct paths starting from the root. Using this representation, the authors perform XML documents classification and clustering, applying methods developed in [63] and [12] respectively. However, the approach is not compared to existing methods which could also be utilized for classification/clustering purposes (e.g., *ED*-based, tag similarity, edge matching or path similarity).

Fast Fourier Transform: An original XML similarity approach, developed in the context of XML document clustering, is presented in [29]. Here, Flesca *et al.* represent the structure of an XML document as a time series (disregarding OLTs), each tag occurrence of an XML element/attribute corresponding to an impulse. Subsequently, they determine the degree of structural similarity between documents by analyzing the frequencies of the Fast Fourier Transform of corresponding time series. The overall complexity of Flesca *et al.*'s approach [29] simplifies to that of the FFT: $O(N \times log(N))$. However, the author in [10] provides an experimental critique of Flesca *et al.*'s FFT method [29]. While it runs faster than a variant of tree *ED*-based approaches, Buttler in [10] concludes that the FFT approach does not offer an accurate measure of similarity. Clustering experiments, conducted on both real and synthetic XML data, show that the FFT method always yields the highest error rates (largest number of mis-clustered documents, i.e. documents put in wrong clusters).

Structural similarity via Entropy: In [40], Helmer introduces a method for measuring the structural similarity between XML documents using entropy (i.e., information distance). The method consists of two main steps. First, the author extracts structural information (tag sequences, edges and paths) from the documents at hand. Second, the structural *informations* concerning each document X and Y are concatenated and then compressed (obtaining $C(X)$. $C(Y)$ and $C(XY)$ as the lengths of the compressed files corresponding to the structural *informations* of documents X, Y and their concatenation respectively). The compressions are hence exploited in computing entropy:

$$Sim_{Entropy}(X, Y) = \frac{C(XY) - Min\{C(X), C(Y)\}}{Max\{C(X), C(Y)\}} \quad \textbf{(4)}$$

The rationale behind this original method is the following: *the more overlap between documents, the better the compression rate will be*, and thus the higher the similarity. The approach is of $O(N+M)$ complexity, where N and M are the respective numbers of elements in the documents being compared. The author compared his method to one of the main *ED*-based XML similarity approaches [61], to the *Fast Fourier Transform* method [29] as well as to a *path similarity* variant [10]. Clustering experiments show that the proposed method produces higher document clustering quality than [29] and [10], and that it's on a par (and in certain test cases better) than the *ED*-based method in [61].

Structural pattern matching: Alternatively to document/document comparison methods, an approach for document/pattern comparison is provided by Sanz *et al.* in [72]. It is dedicated to structural ranked XML querying, searching for a given XML pattern (XML tree) in a document collection. It starts by matching an XML pattern to sub-trees in the XML data tree, taking into account node label similarity. Node labels l_1 and l_2 are considered similar if: l_1 is identical to l_2, l_1 is a synonym of l_2 w.r.t. a given thesaurus, or l_1 is syntactically similar to l_2 w.r.t. to a string matching technique (e.g., string edit distance [48]). In a subsequent phase, the hierarchical structure of nodes is considered to identify, among the possible matches, those that are structurally more similar. Dedicated indexing structures (underlining the label, pre-order, post-order and depth of each node) are utilized for representing patterns and regions in the document collection. Timing results in [72] show that performance is linearly dependent on the size, in number of nodes, of the result set (w.r.t. the considered pattern).

XML/DTD similarity: A method for measuring the structural similarity between an XML document and a DTD grammar is provided by Bertino *et al.* in [6]. The proposed algorithm takes into account the level (i.e. depth) in which the elements occur in the hierarchical structure of the XML and DTD tree representations. Elements at higher levels are considered more relevant, in the comparison process, than those at lower levels. The algorithm also considers element complexity (i.e. the cardinality of the sub-tree rooted at the element) when computing similarity values. The authors state that their approach is of exponential complexity. They show that complexity becomes polynomial ($O(\Gamma^2 \times (N+M))$ where M is the number of nodes – elements/attributes – in the XML document tree, N the number of nodes – elements/attributes as well as ?, *, +, *And*, *Or* operators – in the DTD tree, and Γ the maximum number of edges out coming from a node of the XML document) if the following assumption holds: *In the declaration of an element, two sub-elements with the same tag are forbidden*. The authors also provide a detailed discussion of the possible applications for such an approach, mainly document classification (cf. Section 4).

3.3.2. Structure-and-content XML similarity methods

While different methods to XML similarity, disregarding element/attribute values and focusing on the structural properties of XML data, have been proposed in the literature, many others consider values in their similarity computations. Methods of the latter group target XML documents which are less structurally disparate (they might originate from the same data source, and might even conform to the same grammar), and are mainly developed in the contexts of XML change management, data integration and *structure-and-content* ranked querying.

Leaf node clustering: In the context of XML data integration (cf. Section 4), Liang and Yokota provide in [51] an approximate XML similarity method based on leaf nodes (leaf node values in particular), entitled *LAX* (Leaf-clustering based Approximate XML join algorithm). Following *LAX*, the approximate similarity between two trees is estimated as the mean value of the similarity between their corresponding sub-trees (an algorithm dedicated for segmenting XML documents into independent sub-trees, to be treated via *LAX*, is also provided in [51]). The similarity degree between two sub-trees is determined as the percentage of the number of matched leaf nodes (pairs of leaf nodes that have the same data value) out of the total number of leaf nodes in the sub-trees. The approach is of overall complexity $O(N^2)$ where N is the maximum number of nodes in the XML documents being compared. Experiments in [51] show that *LAX* is, in general, effective in assessing XML documents similarity, w.r.t. to tree *ED*. Nonetheless, the authors state that when large XML documents come to play, i.e. when documents have to be fragmented to fit in main memory, similarity results might not be optimal (Note that following *LAX*, the similarity between two tree documents depends on those of their sub-trees. As a result of fragmentation, sub-trees that share the largest similarities – i.e. matching sub-trees – might not be detected, each group of fragments being treated separately. Hence suboptimal tree comparison results are attained). Recall that the author in [18] provides an approach (*ED*-based) capable of comparing XML documents that are too large to fit in main memory, without affecting the algorithm's optimality. Despite the fact that the approaches target different application domains (the latter being primarily developed for change management purposes), it could have been interesting to compare the two methods.

Document List similarity: In [43], Kade and Heuser develop a method for comparing XML documents as *documents lists*. The comparison process is undertaken in two steps. First, each and every sub-tree of the document tree is traversed, producing for each sub-tree, a string made of the contents (values) of all the sub-tree's leaf nodes merged together. The result is a set of tuples of the form <*path, content*>, one for each node in the XML tree (e.g., the tuple corresponding to node *Department* in Figure 5 is <*Academy/Faculty/Department, 'John Cramer John Takagi James Tailer'*>). This representation of the XML document is called *document list*. The second step of the comparison process consists in comparing the obtained *document lists*, identifying matching nodes (tuples) following their content and label similarities (using string based comparison techniques, e.g., string edit distance [48], as well as path similarities (using path-based comparison methods such as the ones described in the *Path similarity* paragraph above). Nodes having a pair-wise similarity value above a predefined threshold are considers as matching nodes. Note that authors only consider pairs of matching nodes to be significant in computing overall

XML document similarity (Document similarity is computed as the average of all similarity scores between matching nodes). However, in discussing their experimental results, they suggest to relax this criterion so as to decrease document similarity w.r.t. to the number of unmatched nodes, to get more accurate comparison results. Ranking experiments are conducted on (only) synthetically generated documents. The authors explain that they could not find real XML data suitable for their experiments.

Object Description similarity: Weis and Naumann in [86] put forward a method entitled *Dogmatix* for comparing XML elements (and consequently documents) based on their direct values, as well as corresponding parent and children similarities. It is developed in the context of duplicate element detection, i.e., identifying elements that represent the same real world entity, and is geared toward data integration. The method consists of three main steps: i) candidate detection, ii) object description and ii) similarity computation. The first step identifies, in the XML documents being compared, which elements are relevant for comparison, i.e., elements that might describe the same real world entity. Hence, a predefined mapping between the elements of grammars describing the XML documents at hand, and real world entities, is provided as input to the process. The second step comes down to defining the descriptions of those elements, entitled *object descriptions (ODs)*. For a given element e_0, the *object description* comprises of a set of <*name, value*> tuples underlining e_0's value, sibling, children and/or parent data, and are identified using dedicated heuristics and conditions. Heuristics are such as *r-distant descendents*, considering the first *r* elements which depths in the document do not differ more than radius *r* form element e_0's depth itself. For example, the *OD* of element *Department* (cf. Figure 5) following heuristic *r-distance descendents* at radius *r=1* is *{<Professor, John Cramer>, <Student, John Takagi>, <Student, James Tailor>}*. Similar heuristics are proposed for identifying sibling and parent data descriptions. The third step of the comparison framework consists in comparing XML elements based on their *object descriptions*. Textual values are compared using a variation of string edit distance [48]. Overall element similarities are evaluated using a variation of the *IDF* score (*Inverse Document Frequency*) [69], considering the number of matched *OD* tuples (which similarity is above a given threshold) over the total number of tuples in the two *ODs* corresponding to the elements being compared. The similarity between two XML documents is evaluated as that of their root elements. Experimental results in [86] show that *Dogmatix* is effective in identifying real and synthetic duplicate XML elements/documents.

Bayesian Networks: Another interesting approach to duplicate detection is developed in [47]. It considers the complete sub-structure (children and descendents) of each element in the documents at hand (not only the element's children as in [86]). It follows a probabilistic approach, using a Bayesian network to combine the probabilities of children and descendents being duplicates, for a given pair of XML elements in the documents being compared. The similarity between two XML documents corresponds to the probabilities of their root nodes being duplicates. Documents being compared should conform to the same grammar so as to construct the Bayesian network. The latter strictly follows the underlying document grammar. If documents are not well formed w.r.t. to the same grammar, a grammar matching phase should precede the construction of the Bayesian network. The approach's complexity comes down to $O(N^2)$ in the worst case scenario, where N is the maximum number of nodes in the documents being compared. Experimental results in [47] show that the proposed method is, in general, more effective in detecting duplicates, in comparison with *Dogmatix* [86]. The authors however stress on the need for further improvements, particularly concerning the use of *IDF* (*Inverse Document Frequency*), which was proven effective with *Dogmatix*, particularly when the compared documents encompass many elements with dummy or repeated values (For instance, the *actor role* or *production year* element values, in a given XML document describing movies, are not as discriminating in identifying movies, as the movie's *title* value. This can be taken into account using *IDF*, since the same *role*/*year* values could appear in many movies, whereas the *title* value does not).

Pattern matching: An approach for document/pattern comparison, developed in the context of data integration and XML querying, is proposed in [26]. Dorneles *et al.* model XML documents and patterns as ordered labelled trees, and evaluate document/pattern similarity using different metrics dedicated to atomic elements (i.e., leaf nodes in the XML tree) and complex ones (encompassing other elements, i.e., inner nodes in the XML tree) respectively. On one hand, authors consider atomic element metrics to be dependent on the domains of corresponding values (texts, dates, numbers, …) and thus do not detail this issue. On the other hand, they distinguish between complex *collection* and *tuple* element metrics. Following Dorneles *et al.* [26], a *tuple* element is made of different sub-elements (e.g., root node *a* of *Tree C* in Figure 4) whereas a *collection* element encompasses repetitions of the same sub-element (e.g., node *a* of *Tree B* in figure 4). Metrics are provided for both types of complex elements. These are recursively evaluated, exploiting the atomic element metrics, so as to quantify the similarity between the XML pattern and document trees at hand. Note that this method is dedicated to comparing documents and patterns which are fairly similar. The authors state that the satisfactory evaluation of XML element similarities, following their method, requires the compared elements to share the same contexts (their root paths should be identical) and have similar children. Authors in [26] do not compare their method's quality levels to existing approaches, mainly those targeting ranked XML querying (e.g., *IR*-based). That is probably due to the complex nature of such a task, as discussed in Section 3.2.2.

Table 3 depicts the various XML similarity methods discussed above, along with their basic features and application domains.

4. Applications of XML similarity

The use of XML similarity ranges over a wide spectrum of applications which can be classified in four major groups: i) data warehousing, ii) XML classification/clustering, iii) XML data integration, and iv) ranked XML querying.

4.1. Data warehousing: version control and change management

One of the main applications of XML comparison is to provide support for the control of changes in a warehouse of XML documents. In such a context, an *ED*-based measure (via an *edit distance algorithm*, which can also be identified in this context as a *differencing algorithm*, or simply *diff*), i.e. a similarity measure that provides *deltas* (basically *edit scripts*), is required. The *deltas* offer means to change detection and representation between XML documents, and constitute the building blocks for XML versioning [16][17][18][20]. *Deltas* resemble traditional logs in database systems, and similarly to databases, one can find many applications that require access to logs [55].

Versions and querying the past [18][20][55][84]: One may want to view or access a version of a particular document, (part of) a Web site, or the results of a continuous query. This is the standard use of versions, namely recording history (i.e. obtain an old version of an XML document). Later, one might want to ask a query about the past (e.g. ask for the value of some XML element at some previous time) and to query changes (e.g. ask for the list of items recently introduced in a document/catalog). Since the *deltas* can be stored as XML documents, such queries become regular queries over documents.

Learning about changes: The edit distance algorithm constructs a possible description of the changes. It allows to find, mark-up, and browse changes between two or more versions of a document and also to update the old versions of the document. This is in the spirit, for instance, of the Information and Content Exchange, ICE[1] [85]. Also, different users may modify the same XML document off-line, and later want to synchronize their respective versions. The edit distance algorithm could be used to detect and describe the modifications in order to detect conflicts and solve some of them [20].

Monitoring changes: In the *Xyleme* project for instance [20], [55], [60], monitoring changes serves as the first facet to query subscription and notification systems. The authors implement a subscription system [60] that allows detecting changes of interest in XML documents, e.g. that a new product has been added to a catalogue. At the time the *delta* is computed (e.g. the edit distance algorithm is executed), the system verifies for each atomic change whether this change is monitored by some subscription (e.g. the insertion of a new item in the XML document, a deletion of a given item, etc.). Note that this is relevant to ICE which also provides a protocol for notification.

Archiving: Archiving is straightforward in this context. It suffices to store the sequence of *deltas* before a certain date to archive corresponding XML data [55].

Mirroring: XML comparison, via edit distance algorithms, can also be used to reduce the amount of data transmitted over a network in mirroring applications [18]. Popular Web and FTP servers often have dozens of mirror sites around the world. Changes made to the master server need to be propagated to the mirror sites. Ideally, the users or programs making changes would keep a record of exactly what data was updated. However, in practice, due to the autonomous and loosely organized nature of such sites, there is no reliable record of changes [18]. Therefore, efficient mirroring requires *diff* algorithms that compute and propagate only the difference between the data version at the server and that at a mirror site.

4.2. XML classification and clustering

Among the main uses of XML similarity/comparison are the classification and clustering of XML documents.

XML classification: XML similarity/comparison enables the classification of XML documents gathered from the web against a set of XMl grammars (DTDs or XML schemas) declared in an XML database. The scenario provided by Bertino *et al.* [6] comprises a number of heterogeneous XML databases that exchange documents among each other, each database storing and indexing the local documents according to a set of predefined DTDs. Consequently, XML documents introduced in a given database are matched, via an XML structural similarity method, against the local DTDs. Note that *matching*, in such an application, can be undertaken using an XML document/DTD comparison method (like the one proposed in [6] for measuring the similarity between an XML document and a DTD definition) or via an XML document/document comparison method (e.g., one of the methods described previously for comparing two XML documents). Following the latter strategy, the DTD will be exploited as a generator of XML document structures (set of possible document structures valid for the DTD is considered). Then, for each document structure, algorithms for measuring the structural similarity between XML documents, e.g. [18], [23], [61], can be applied. The matching resulting in the highest similarity value can be

[1] In the context of electronic commerce, the ICE is a standard that supports exchanging information about changes of a set of web pages. It is also based on *deltas* and snapshots of the data [85].

considered as the best candidate, the corresponding similarity value being considered as the structural similarity score between the document and the DTD). In such an application, a similarity threshold should be identified (by the user or the system), underlining the minimal degree of similarity required to bind an XML document to a DTD [6]. The DTD, for which the similarity degree is highest, and above the specified threshold, is selected. Thus, the XML document at hand is accepted as valid for that DTD. When the similarity degree is below the threshold, for all DTDs in the XML database, the XML document is considered *unclassified* and is stored in a repository of unclassified documents. As a result, none of the protection, indexing and retrieval facilities specified at DTD level can be applied to such documents [6].

XML clustering: Grouping similar XML documents together can improve data storage indexing [76], and thus positively affect the retrieval process. For instance, if two documents/elements are similar, it is likely that they both either satisfy or not a given query. Therefore, when grouped together, similar documents/elements would be much easier to retrieve than when scattered at different locations in the storage device [49].

XML clustering can also play a major role in effective DTD extraction [33]. A lot of XML documents found on the web are heterogeneous and lack predefined grammars (DTDs or schemas). Nonetheless, having knowledge of the grammar, for a set of XML documents, can be valuable for the protection, indexing, querying and retrieval of these documents [6], [61]. Just as schemas are necessary in traditional DBMS, the same is true for DTDs and XML databases. Given a collection of heterogeneous XML documents, constructing a single DTD for all these documents would lead to a far too general definition, which would not be of much of use. However, when structurally similar documents are clustered together before the DTD extraction process, a more accurate and specific DTD will be constructed for documents in each cluster [61].

Clustering can also be critical in information extraction. Current information extraction methods either implicitly or explicitly depend on the structural features of documents [10]. Based on structural clustering, it would be much easier to automatically identify the sets of XML documents that are useful to information extraction algorithms, and that would produce meaningful results.

4.3. Data integration

XML similarity/comparison is also a central issue in data integration. One of the main features of XML is that it can represent different kinds of data from different data sources, mainly on the web. Nonetheless, XML documents from different data sources might contain nearly or exactly the same information but might be constructed using different structures. In addition, even if two documents express similar contents, each of them may have some extra information w.r.t. to the other. Thus, one needs an effective XML similarity measure in order to integrate such data sources, so that the user can conveniently access and acquire more complete information [38], [50], [51].

More precisely, the problem of integrating two XML data sources, from a similarity/comparison point of view, comes down to performing an *approximate join* between these sources using a predefined XML similarity measure (most likely a tree edit distance based measure). Given two XML sources, S_1 and S_2, a similarity threshold s, and a function $Sim(d_1, d_2)$ that assesses the similarity between two documents $d_1 \in S_1$ and $d_2 \in S_2$, the *approximate join* between data sources S_1 and S_2 reports in the output all pairs of documents $(d_1, d_2) \in S_1 \times S_2$ such that $Sim(d_1, d_2) \geq s$ [37][38]. Subsequently, integrating the identified pairs of documents to form unified views of the data can be undertaken.

4.4. Ranked XML querying

With the increasing use of XML, specifically on the web, efficient retrieval of XML documents becomes more and more important [31]. The database community has proposed several languages for querying XML, including XML-QL [25], XQL [67] and XQuery [15]. However, these languages are based on exact matching and do not support ranked queries via textual/structural similarity. Therefore, several attempts have been made to extend these query languages in order to support ranked results, which is where XML similarity techniques come to play.

While most approaches in this framework are based on extensions of the vector space model, the query model used varies with each approach. In [31], the authors extend XQL, introducing the query language XIRQL which incorporates the notions of term weights and vague predicates. In [13], Carmel *et al.* avoid defining a new XML query language and allow the users to express their information needs as simple XML fragments (i.e. parts of XML documents). The underlying idea is to give less control to the user when formulating queries, and to focus most of the logic in the ranking mechanism in order to best meet the user's needs (similarly to *free text* query mechanisms in traditional *IR*). In [75], Schlieder and Meuss support structured queries, i.e. queries are labeled trees, and thus give further attention to XML structure in the retrieval process. Another study by Schlieder [74] introduces a simple query language entitled approXQL that supports hierarchical Boolean-connected query patterns. The interpretation of approXQL queries is founded on cost-based query transformations where queries/documents are modeled as labeled trees. Thus, a tree edit distance (approximate tree matching) variation is used to compute the cost of a sequence of transformations between a query and the data and is used to rank the results. Approaches comparable to [74] are provided in [4], [56]. In these more recent works, the authors make use of structural relaxation on XPath queries, defining specific relaxation operations (*edge generalization* – i.e. transforming a parent/child edge to an ancestor/descendent one, *leaf node deletion* …) and

dedicated scoring functions to enable ranked query answering. An approximate answer to the original XPath user query is none other than an exact match to one of its relaxed queries, its score accounting for the corresponding relaxation process (i.e. the score resulting form applying the relaxation operations to obtain the relaxed query at hand). For instance, the document tree in Figure 5 is an approximate answer to the XPath query *//Academy[./Faculty[./Section]]* with a score corresponding to deleting leaf node *Section* so as to obtain the relaxed query *// Academy [/Faculty]*. Various scoring and ranking schemes are proposed.

In [19], Chinenyanga and Kushmerick try to adapt existing ranking capabilities in relational database systems to XML retrieval. The authors put forward ELIXIR, a language for XML information retrieval that extends XML-QL with a textual similarity operator. The corresponding query answering algorithm rewrites the original ELIXIR queries into a series of intermediate relational data, and makes use of WHIRL[1] [21], [22] to efficiently evaluate the similarity operator on this intermediate data, subsequently yielding ranked XML results. Similarly to [19], Theobald and Weikum [81] introduce an XML query language, XXL, extending XML-QL with a similarity operator. Note that XXL's similarity operator can be applied to element/attribute names as well as to element/attribute values whereas ELIXIR [19] is bound to element/attribute values. The corresponding query processor makes use of the Oracle Inter-Media thesaurus while computing similarity. In [8], Bremer and Gertz motivate the introduction of a *rank* operator in the XQuery syntax (the resulting query language is identified as XQuery/IR) enabling the user to choose the similarity/comparison method to be utilized in the process (the authors do not specify the underlying *IR* technique to be used).

On one hand, XML data warehousing, data integration and classification/clustering applications require relatively accurate XML similarity methods so as to produce better results (more accurate change detection, more complete data integration and higher quality classification/clustering respectively). Hence, approaches in these application areas are generally *ED*-based (fined-grained). Domain specific methods (e.g., edge matching, path similarity…, some trying to approximate more complex *ED*-based methods) have been proposed (cf. Section 3.3). On the other hand, most ranked XML querying studies tend to favour performance on accuracy, aiming to produce good enough results in reasonable time (instead of trying to generate perfectly correct results). Therefore, most methods in this application domain are *IR*-based (coarse-grained).

Nevertheless, a few ranked XML querying approaches have tried to close this gap by exploiting variants of *ED* [74] and other techniques based on path similarity [4], [56]. Thus, adapting *ED*-based methods (or other edge-based, path-based, …, techniques) to search and retrieve XML data or, on the other hand, adapting *IR*-based methods to data-warehousing, data integration, classification and clustering applications could yield interesting results (Note that the idea of utilizing *ED* computations in a ranked querying system, for instance, is not novel. It was introduced by Shasha and Zhang in [77] in the context of generic tree structures querying).

[1] WHIRL is an information retrieval query language dedicated to relational data. It includes a textual similarity metric and provides ranked similarity results [21], [22].

5. Discussions and future research directions

While substantial work has been conducted around the XML similarity problem, various issues regarding the efficiency, performance and potential applications of XML comparison approaches are yet to be tackled. In the remainder of this section, we present some of these issues. First, we discuss several limitations of current approaches, w.r.t. the structural characteristics of XML data, while comparing XML documents. After, we present a glimpse on one of the emergent problems related to XML similarity: the combination of structural and semantic similarity assessment while comparing XML data, which is being investigated in both *ED* and *IR*-based approaches. To conclude this section, we discuss the usefulness of XML grammars (DTDs or XML schemas) in developing improved XML comparison methods.

5.1. XML structural similarity

As shown previously, a range of algorithms for comparing highly structured XML documents have been proposed in the literature. A thorough investigation of the most recent and efficient XML structural similarity approaches led us to pinpoint certain cases where the corresponding comparison outcome is inaccurate.

Usually, XML documents can encompass many optional and repeated elements [61]. Such elements induce recurring sub-trees of similar or identical structures. As a result, algorithms for comparing XML document trees should be aware of such repetitions/resemblances so as to efficiently assess structural similarity.

Note that in the following, we mainly focus our discussions on *ED*-based XML comparison algorithms since they target rigorously structured data, and thus are more fine-grained w.r.t. *IR*-based methods. Nonetheless, the limitations pinpointed consequently transitively cover *IR*-based XML similarity methods, and others, as well.

5.1.1. Undetected Sub-tree Similarities

Our examination of the approaches provided in [18], [61], [23] led us to identify certain cases where sub-tree structural similarities are disregarded (cf. Figure 6):

- Similarity between trees *A*/*D* (sub-trees A_1 and D_2) in comparison with *A*/*E*.

- Similarity between trees *F*/*G* (sub-trees F_1 and G_2) relatively to *F*/*H*.
- Similarity between trees *F*/*I* (sub-tree F_1 and tree *I*) in comparison with *F*/*J*.

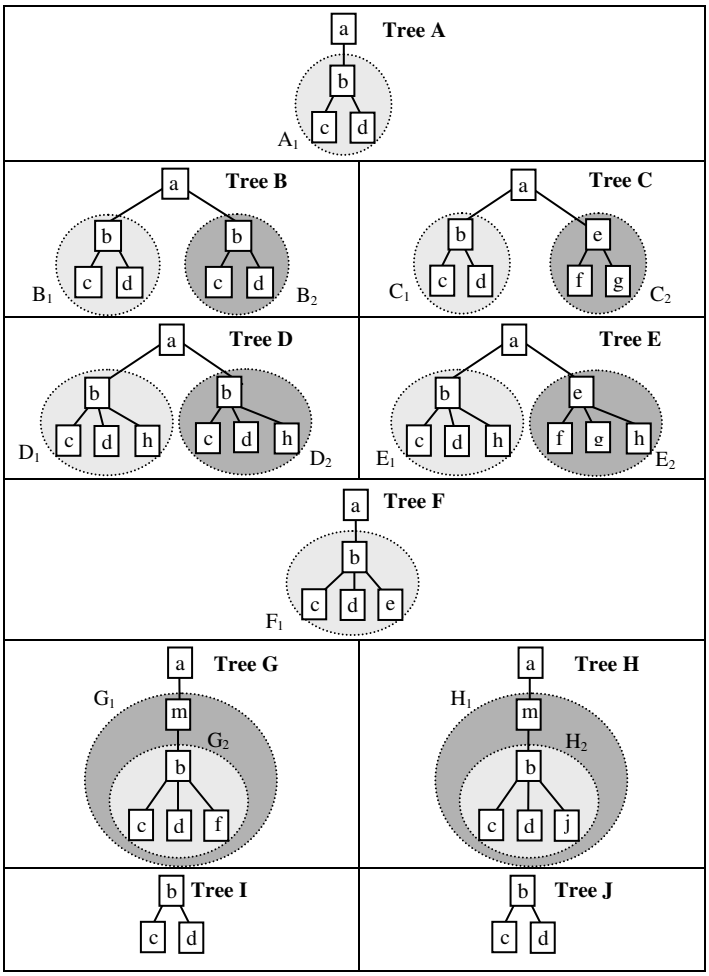


Fig. 6 - Sample XML trees.

Nierman and Jagadish in [61] make use of the *contained in* relation between trees to capture sub-tree similarities, such as with the *A*/*B* and *A*/*C* case mentioned in Section 3.1.2 (repetition of sub-tree B_1). Nonetheless, when the containment relation is not fulfilled, certain structural similarities are ignored. Consider, for instance, trees *A* and *D* in Figure 6. Since D_2 is not contained in *A*, it is inserted *via* four edit operations instead of one (insert tree), while transforming *A* to *D*, ignoring the fact that part of D_2 (sub-tree of nodes *b*, *c*, *d*) is identical to A_1. Therefore, equal distances are obtained when comparing trees *A*/*D* and *A*/*E*, disregarding *A*/*D*'s structural resemblances (here, we assume the general case where atomic insertion/deletion operations are of unit costs, =1):

- $Dist(A, D) = Cost_{Ins}(h) + Cost_{Ins}(b) + Cost_{Ins}(c) + Cost_{Ins}(d) + Cost_{Ins}(h) = 1 + 4 = 5$
- $Dist(A, E) = Cost_{Ins}(h) + Cost_{Ins}(e) + Cost_{Ins}(f) + Cost_{Ins}(g) + Cost_{Ins}(h) = 1 + 4 = 5$

Likewise for the *D* to *A* transformation (tree D_2 will not be deleted via a single delete tree operation since it is not contained in the destination tree *A*), achieving *Dist(D, A) = Dist(E, A) = 5*. Other types of sub-tree structural similarities that are disregarded by Nierman and Jagadish's approach [61] (and likewise missed in [23], [18]) can be identified when comparing trees *F*/*G* and *F*/*H*, as well as *F*/*I* and *F*/*J*. The *F*, *G*, *H* case is different than its predecessor (the *A*, *D*, *F* case) in that the sub-trees sharing structural similarities (F_1 and G_2) occur at different depths (whereas with *A*/*D*, A_1 and D_2 are at the same depth). On the other hand, the *F*, *I*, *J* case differs from the previous ones since structural similarities occur, not only among sub-trees, but also at the sub-tree/tree level (e.g. between sub-tree F_1 and tree *I*).

Note that in [23], Dalamagas *et al.* complement their edit distance algorithm with a repetition/nesting reduction process, summarizing the XML documents prior to the comparison phase. Such a reduction pre-processing transforms, for instance, tree *B* to *A* (repetition of sub-tree B_1) thus yielding *Dist(A, B) = 0* which is not accurate (tree *A* is obviously different than *B*). While it might be useful for structural clustering tasks, the reduction process yields inaccurate comparison results in the general case, which is why it is disregarded in our discussion. As for Dalamagas *et al.*'s *ED* algorithm [23], it yields distance values identical to the ones returned by Nierman and Jagadish's process [61] in the above examples. Recall that the algorithm in [23] is a specialized version of that developed in [61] where tree insertion/deletion costs are computed as the sum of the costs of inserting/deleting all individual nodes in the considered trees.

5.1.2. The Special case of leaf node sub-trees

In addition, none of the approaches mentioned above is able to effectively compare documents made of repeating leaf node sub-trees. For example, following [18][23][61] the same structural similarity value is obtained when comparing document *K*, of Figure 7, to documents *L* and *M*, *Sim(K, L) = Sim(K, M) = 0.5*, having *Dist(K, L) = Dist(K, M) = 1*.

- $Dist(K, L) = Cost_{Ins}(b) = 1$
- $Dist(K, M) = Cost_{Ins}(c) = 1$

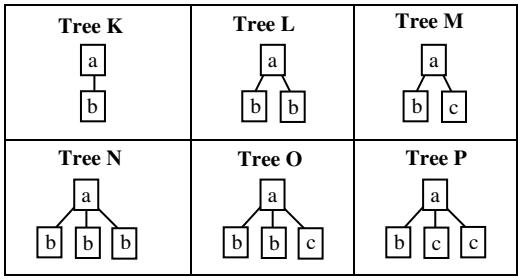


Fig. 7 - XML documents consisting of leaf node sub-trees.

However, one can realize that document trees *K* and *L* are more similar than *K* and *M*, node *b* of tree *K* appearing twice in tree *L*, and only once in *M*. Likewise for *K/N* with respect to *K/O* and *K/P*. Identical distances are attained when comparing document trees *K/N, K/O* and *K/P, Dist(K, N)=Dist(K, O)= Dist(K, P)=2,* despite the fact that node *b* is repeated three times in *N*, twice in *O* and only appears once in *P*.

- $Dist(K, N) = Cost_{Ins}(b) + Cost_{Ins}(b) = 2$
- $Dist(K, O) = Cost_{Ins}(b) + Cost_{Ins}(c) = 2$
- $Dist(K, P) = Cost_{Ins}(c) + Cost_{Ins}(d) = 2$

We explicitly mention the case of leaf node repetitions since:

- Leaf nodes are a special kind of sub-trees: *single node sub-trees*. Therefore, the issue of sub-tree resemblances and repetitions should logically cover leaf nodes, so as to attain a more complete XML similarity approach.
- Leaf node repetitions are usually as frequent as substructure repetitions (i.e. non-leaf node sub-tree repetitions) in XML documents.

Detecting leaf node repetitions is spontaneous in the XML context, and would help increase the discriminative power of XML comparison methods, as shown in the examples of Figure 7.

5.2. Semantic similarity

Combining structural and semantic XML similarity is one of the hot topics recently being investigated. Most similarity approaches in the literature focus exclusively on the structure of documents, ignoring the semantics involved. However, in the field of Information Retrieval, estimating semantic similarity between web pages is of key importance to improving search results [53]. In order to stress the need for semantic relatedness assessment in XML document comparison, we report from [79] the examples in Figure 8.

```
<?XML?>
<Academy>
  <Department>
   <Laboratory>
    <Professor></Professor>
    <Student> </Student>
   </Laboratory>
  </Department>
</Academy>
```

```
<?XML?>
<College>
 <Department>
  <Laboratory>
   <Lecturer></Lecturer>
  </Laboratory>
 </Department>
</College>
```

```
<?XML?>
<Factory>
 <Department>
  <Laboratory>
   <Supervisor></Supervisor>
  </Laboratory>
 </Department>
</Factory>
```

Fig. 8 - Examples of XML documents

Using classical ED computations, the same structural similarity value is obtained when document *A* is compared to documents *B* and *C* [79]. However, despite having similar structural characteristics, one can obviously recognize that sample document *A* shares more semantic characteristics with document *B* than with *C*. For example, in Figure 8, pairs *Academy-College* and *Professor-Lecturer*, from documents *A* and *B*, are semantically similar while *Academy-Factory* and *Professor-Supervisor*, from documents *A* and *C*, are semantically different. Therefore, taking into account the semantic factor in XML similarity computations would obviously amend similarity results.

In recent years, there have been a few attempts to integrate semantic and structural similarity assessment in the XML comparison process. One of the early approaches to propose such a method is [81]. The authors here make use of a textual similarity operator and utilize Oracle's Inter-Media text retrieval system to improve XML similarity search. In a recent extension of [81] provided in [73], Schenkel *et al.* propose a generic ontological model, built on WordNet[1], to account for semantic similarity (instead of utilizing Oracle Inter-Media). On the other hand, the authors in [6], [72] identify the need to support tag similarity (synonyms and stems[2]) instead of tag syntactic equality while comparing XML documents. In [79], the authors study the XML semantic similarity issue in more detail. They consider the various semantic relations encompassed in a given reference taxonomy/ontology (e.g. WordNet) while comparing XML documents. They introduce a combined structural/semantic XML similarity approach integrating *IR* semantic similarity assessment in a traditional ED algorithm [18].

Nonetheless, the semantic/structural similarity problem is far from solved. The vast differences between the proposed approaches suggest that semantic similarity could be integrated in multiple ways while comparing XML data. In addition, the semantic complexity issue, which is due to accessing the taxonomy/ontology considered, is currently an open problem. Experimental results in [79] confirm the positive impact of semantic meaning on XML similarity values, while underlining its heavy impact regarding timing complexity. Therefore, this emergent topic is likely to be thoroughly investigated in the following years.

5.3. Exploiting XML grammars

Another possible future research direction would be to explore the use of existing XML similarity methods to compare, not only the skeletons of XML documents (element/attribute labels) but also their information content (element/attribute values). In current approaches, when element/attribute values are considered in the comparison process [74], [90], they are treated as strings (i.e. of data type *String*) which is not always the case. Values could be of *Decimal*, *Boolean*, *String*, *Date*… types. For each data-type, a different method should be utilized to compute similarity. Therefore, in such a framework, XML Schemas might have to be integrated in the comparison process,

[1] WordNet is an online lexical reference system (taxonomy), developed at Princeton University NJ USA, where nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing a lexical concept [58] (http://www.cogsi.princeton.edu/cgi-bin/webwn).

[2] *Stems* designate the morphological variants of a term: an acronym and its expansions, a singular term and its plural, …

schemas underlining element/attribute data types[1] which are required to compare corresponding element/attribute values. A direct application of such an approach would be the development of a sophisticated XML query and retrieval system.

It would also be interesting to tackle the XML/Grammar comparison issue, introduced in [6]. Since [6], [80] are (to our knowledge) the only works that cover this problem, it seems interesting to look into that topic. As shown in [6], measuring the structural similarity between an XML document and a DTD has various applications, including XML documents classification, DTD structure evolution, the evaluation of structural queries, the selective dissemination of XML documents as well as the protection of documents.

6. Conclusion

In the past few years, XML has been established as the *de facto* standard for web publishing [84], attracting growing attention in database, information retrieval, and more recently multimedia related research (XML is being increasingly used for describing complex objects, e.g. multimedia information, such as MPEG-7[2], SVG[3], X3D[4], etc.).

In this paper, we gave an overview of existing research related to XML similarity. The wide range of diverse methods proposed in the literature were roughly organized into three major groups: i) *ED*-based (Edit Distance), ii) *IR*-based (Information Retrieval), and iii) various other context and application specific techniques to XML comparison (some methods in this group are approximations of more complex existing methods – mainly *ED*-based). While *IR*-based methods target XML search and retrieval (especially for loosely structured *document-centric* XML), *ED*-based techniques seem to focus more on the structural aspect of XML (rigorously structured *data-centric* view) and are primarily utilized for classification/clustering and data warehousing purposes. We detailed the possible applications of XML comparison processes in various fields, ranging over data warehousing, data integration, classification/clustering and ranked XML querying. In addition, we discussed some possible future research directions, covering XML structural and semantic similarity, as well as the exploitation of XML grammars in developing improved XML comparison methods.

To conclude, note that *ED*-based methods for comparing XML documents have been thoroughly studied in the past decade, exploiting and extending achievements of the combinatorial pattern matching community in tree *ED* algorithms and related processes. However, they are yet to be further improved and perfected as shown in the previous section. On the other hand, *IR*-based XML similarity is recently gaining increasing importance, especially through the INEX evaluation campaigns sponsored and organized by the *IR* community. And since most *IR*-based methods are "more or less" heuristic, they are incessantly discussed, which presents an overwhelming motivation to venture in the field.

We believe that the unified presentation of XML similarity in this paper will facilitate further research on the subject.

[1] XML Schemas, like DTDs, provide a means for defining the grammar of a set of XML documents. However, schemas enable a thorough management of data-types (19 different data-types are supported, the user being able to derive new data-types based on the ones that are built-in), which is very restricted in DTDs. XML Schema Part 2: Datatypes Second Edition. http://www.w3.org/TR/xmlschema-2/.

[2] Moving Pictures Experts Group, MPEG-7 http://www.chiariglione.org/mpeg/standards/mpeg-7/.

[3] WWW Consortium, SVG, http://www.w3.org/Graphics/SVG/.

[4] Web 3D, X3D, http://www.web3d.org/x3d/.

Tab. 1 - Characteristics of existing *ED*-based XML similarity approaches

	Approach	XML data targeted[1]	Features	Complexity	Applications
Edit Distance methods	Tai [78]	------	– First non-exponential tree *ED* algorithm. – Insert/delete node anywhere in the tree, and update node operations.	$O(\|T_1\|\times\|T_2\|\times depth(T_1)^2\times depth(T_2)^2)$ (cf. notations in following page)	------
	Zhang and Shasha [89]	------	– Insert/delete node anywhere in the tree, and update node operations.	$O(e^2 \times \|T_1\| \times min(\|T_1\|, \|T_2\|))$	------
	Shasha and Zhang [77]	------	– Insert/delete node anywhere in the tree, and update node operations.	$O(\|T_1\|\times\|T_2\|\times depth(T_1) \times depth(T_2))$	Querying tree structures (approximate tree by example queries)
	Chawathe *et al.* [16]	Document/Document Structure-and-content	– Insert/delete leaf node, update node, move node (sub-tree). – The algorithm should match specific criterions and assumptions without which the results attained would be suboptimal.	$O(n\times e + e^2)$	Version control and change management of semi-structured data
	Cobéna *et al.* [20]	Document/Document Structure-and-content	– Insert/delete leaf node, update node, move node (sub-tree). – Some sets of move operations might not be optimal.	$O(N \times log(N)$	XML Version control and change management
	Chawathe [18]	Document/Document Structure-only	– Insert/delete leaf node, update node. – Algorithm extended for external-memory computations.	$O(N^2)$	Version control and change management of semi-structured data
	Nierman and Jagadish [61]	Document/Document Structure-only	– Insert/delete leaf node, update node, insert/delete sub-tree. – Outperforms [Chawathe 1999]'s algorithm, which in turn yields better structural clustering results than [Zhang and Shasha 1989]'s algorithm.	$O(N^2)$	XML Structural clustering
	Dalamagas *et al.* [23]	Document/Document Structure-only	– Insert/delete leaf node, update node, insert/delete sub-tree. – Outperforms [Chawathe 1999]'s algorithm.	$O(N^2)$	XML Structural clustering
	Tekli et al. [80]	Document/Grammar (DTD) Structure-only	– Evaluating structural similarity between XML documents and DTD grammars.	$O(N^3)$	XML structural classification

Tab. 2 - Characteristics of existing *IR*-based XML similarity approaches

	Approach	XML data targeted[2]	Features	Applications
Basic XML Information Retrieval methods	Fuhr and Großjohann [31]	Document/Query (i.e., Document or pattern) Structure-and-content	– Defining indexing nodes. – Computing TF-IDF scores locally. – Augmenting weights w.r.t. the XML structure.	Ranked XML querying
	Chinenyanga and Kushmerick [19]	Document/Query Structure-and-content	– Makes use of existing raking capabilities in relational database systems using WHIRL. – Utilizes classical TF-IDF ranking.	Ranked XML querying
	Grabs and Schek [36]	Document/Query Structure-and-content	– Builds on [Fuhr and Großjohann 2001]. – Introducing the notions of *single category retrieval* and *multi-category retrieval.* – Flexible retrieval: users specify at query time the scope of their queries.	Ranked XML querying
	Carmel *et al.* [13]	Document/Query Structure-and-content	– Considers the context of appearance (the root path) of a term in computing TF-IDF scores. – Relaxing the cosine measure for comparing query/document vectors by accounting for context resemblance.	Ranked XML querying
	Schlieder and Meuss [75]	Document/Query Structure-and-content	– Structural terms (labeled trees) are the basic indexing units. – Flexible retrieval: *logical XML document*	Ranked XML querying
	Amer-Yahia *et al.* [4]	Document/Query Structure-and-content	– XPath query relaxation. – Defining specific relaxations operations. – Scoring answers w.r.t. query relaxation process.	Ranked XML querying
	Pokorny and Rejlek [62]	Document/Query Structure-and-content	– Using matrixes instead of vectors to represent XML documents and queries – The distribution of a term is described w.r.t. its structural distribution in the XMLdocument collection.	Ranked XML querying

[1] Methods in this category target rigorously structured (data-centric) XML
[2] Methods in this category target loosely structured (document-centric) XML

Tab. 3 – Other (context and application specific) methods to XML similarity

	Approach	XML data targeted	Features	Complexity	Applications
Other XML similarity methods	Buttler [10]	Document/Document Structure-only	– Tag similarity. – Document structure is completely ignored.	$O(N)$	XML structural clustering
	Kriegel and Schönauer [45]	Document/Document Structure-only	– Edge matching (matching the edges connecting two XML nodes). – Lower bound of the *ED* techniques (less accurate).	$O(N^3)$	XML structural ranked querying
	Lian *et al.* [49]	Document/Document Structure-only	– Edge matching (matching the edges connecting two XML nodes). – Effective and efficient w.r.t. *ED*.	$O(N^2)$, *simplifies to* $O(k\times N)$ *where k is a small multiple of N*	XML structural clustering
	Rafiei *et al.* [64]	Document/Document Structure-only	– Path similarity (computing the number of common paths between two XML documents). – Effective w.r.t. *ED*.	$O(p\times l^2)$	XML structural clustering
	Joshi *et al.* [42]	Document/Document Structure-only	– XPath similarity – Effective w.r.t. classic path similarity	$O(dp^2)$	XML structural clustering
	Flesca *et al.* [29]	Document/Document Structure-only	– XML documents represented as time. series and compared via FFT – Less accurate than *ED* [Buttler 04].	$O(N \times log(N))$	XML structural clustering
	Helmer [40]	Document/Document Structure-only	– Concatenation and compression of XML document structural properties – Comparison using Entropy	$O(2\times N)$	XML structural clustering
	Sanz *et al.* [72]	Document/Pattern Structure-only	– Exploits dedicated indexing structures to compare XML tree patterns.	$O(K\times P\times Tr)$	XML structural ranked querying
	Bertino *et al.* [6]	Document/Grammar (DTD) Structure-only	– Evaluating structural similarity between XML documents and DTD grammars.	$O(\Gamma^2\times (M+R))$	XML classification, query processing
	Liang and Yokota [51]	Document/Document Structure-and-content	– Leaf node similarity. – Yields suboptimal results when XML documents are too large to fit in main memory.	$O(N^2)$	XML data integration
	Kade and Heuser [43]	Document/Document Structure-and-content	– Documents are transformed into lists of path/content tuples – Comparing document lists via string and path similarity techniques	------	XML data integration
	Weis and Naumann [86]	Document/Document Structure-and-content	– Comparing elements describing the same real world entities – Exploits heuristics in identifying element descriptions	------	XML data integration
	Leiyao *et al.* [47]	Document/Document Structure-and-content	– Exploits a Bayesian network to combine the probabilities of children and descendents being duplicates (similarity higher than threshold)	$O(N^2)$	XML data integration
	Dorneles *et al.* [26]	Document/Pattern Structure-and-content	– Metrics for comparing *tuple* and *collection* complex elements – Comparing elements of the same context (same root path)	------	XML data integration and XML ranked querying

$|T_1|$, $|T_2|$: cardinalities of trees T_1 and T_2
e: weighted edit distance (cf. Section 3.1.2)
N: maximum number of nodes in the trees being compared
n: is the total number of leaf nodes in the trees being compared
p: number of paths
dp: number of distinct paths
l: length of the longest path in an XML document (in number of nodes)
P: cardinality of XML tree pattern
K: maximum size (number of vertices) of a level in the pattern P
Tr: cardinality of the target XML tree
M: is the number of nodes – elements/attributes – in the XML document tree
R: the number of nodes – elements/attributes as well as ?, *, +, *And*, *Or* operators – in the DTD tree
Γ: the maximum number of edges out coming from a node of the XML document

Fig. 9 – Notations corresponding to the complexity formulations in Tables 1, 2 and 3.

REFERENCES

[1] R. Agrawal, C. Faloutsos and A.N. Swami. Efficient Similarity Search in Sequence Databases. In *Proceedings of the 4th International Conference on the Foundations of Data Organization and Algorithms (FODO'93)*, Springer Verlag, (1993) pp. 69-165.

[2] A. Aho, D. Hirschberg, and J. Ullman. Bounds on the Complexity of the Longest Common Subsequence Problem. *Association for Computing Machinery, 23, 1,* (1976) pp.1-12.

[3] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), pp. 357-389.

[4] S. Amer-Yahia, L.K.S. Lakshmanan and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proceedings of ACM SIGMOD*, (2004) pp. 83-94.

[5] V.N. Anh and A. Moffat. Compression and an IR Approach to XML Retrieval. In *Proceedings of the Workshop of the Initiative for the Evaluation of XML Retrieval (INEX'02)*, Germany, (2002) pp. 99-104.

[6] E. Bertino, G. Guerrini and M. Mesiti. A Matching Algorithm for Measuring the Structural Similarity between an XML Documents and a DTD and its Applications, *Elsevier Computer Science, 29*, (2004) pp. 23-46.

[7] M. Boughanem. Introduction to Information Retrieval, In *Proceedings of EARIA 06* (*Ecole d'Automne en Recherche d'Information et Application*), *Chapter 1* (2006).

[8] J.-M. Bremer And M. Gertz. XQuery/IR: Integrating XML Document and Data Retrieval. In *Proceedings of the 5th ACM SIGMOD International Workshop on the Web and Databases (WebDB'02)*, (2002) pp. 1-6.

[9] A. Z. Broder. On the Resemblance and Containment of Documents. In *Proceedings of Compression and Complexity of SEQUENCES*, (1997) pp. 21-29.

[10] D. Buttler. A Short Survey of Document Structure Similarity Algorithms. In *Proceedings of the 5th International Conference on internet Computing*, USA, (2004) pp. 3-9.

[11] L. Candillier, I. Tellier and F. Torre. Transforming XML Trees for Efficient Classification and Clustering. In *Proceedings of the Workshop of the Initiative for the Evaluation of XML Retrieval (INEX'05)*, (2005) pp. 469-480.

[12] L. Candillier, I. Tellier. F. Torre and O. Bouquet. SSC: Statistical Clustering. In *Perner P. and Imiya A. eds.: 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*, Volume LNAI 3587 of LNCS, (2005) pp. 100-109.

[13] D. Carmel, N. Efraty, G.M. Landau, Y.S. Maarek and Y. Mass. An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. In *Proceedings of the ACM SIGIR'02 Workshop on XML and Information Retrieval*, (2002) pp. 14-25.

[14] D. Chamberlin, F. Fankhauser, M. Marchiori And J. Robie. XML Query Requirements, (2000) http://www.w3.org/TR/xmlquery-req.

[15] D. Chamberlin, D. Florescu, J. Robie, J. Simeon and M. Stefanescu. XQuery : A Query Language for XML (2001) http://www.w3.org/TR/2001/WD-xquery-20010215

[16] S. Chawathe, A. Rajaraman, H. Garcia-Molina and J. Widom. Change Detection in Hierarchically Structured Information. In *Proceedings ACM SIGMOD*, Canada, (1996) pp. 26-37.

[17] S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of ACM SIGMOD*, (1997) pp. 26–37.

[18] S. Chawathe. Comparing Hierarchical Data in External Memory. In *Proceedings of the VLDB Conference,* (1999) pp. 90-101.

[19] T.T. Chinenyanga and N. Kushmerick. An Expressive and Efficient Language for XML Information Retrieval. *Journal of the American Society for Information Science 53(6)*, (2002) pp. 438-453.

[20] G. Cobéna, S. Abiteboul and A. Marian. Detecting Changes in XML Documents. In *Proceedings of the IEEE International Conference on Data Engineering*, (2002) pp. 41-52.

[21] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In *Proceedings of ACM SIGMOD*, (1998) pp. 291-211.

[22] W. Cohen. WHIRL: A Word-Based Information Representation Language. *Journal of Artificial Intelligence, 118,* (2000) pp. 163-196.

[23] T. Dalamagas, T. Cheng, K. Winkel, and T. Sellis. A methodology for clustering XML documents by structure. *Information Systems 31, 3,* (2006) pp.187-228.

[24] S. Deerwester, S. Dumais, G. Furnas, T. Landauer and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, *41(6),* (1990) pp. 391–407.

[25] A. Deutsch, M. Fernandez, D. Florescu, A. Levy and D. Suciu. XML-QL: A Query Language for XML. *Computer Networks 31(11-16)*, (1999) pp. 1155-1169.

[26] C.F. Dorneles, C.A. Heuser, A.E.N. Lima, A.S. da Silva and E.S. de Moura. Measuring Similarity between Collections of Values. In *Proceedings of the ACM international Workshop on Web Information and Data Management*, USA, (2004) pp. 56-63.

[27] A. Doucet, L. Aunimo, M. Lehtonen and R. Petit. Accurate Retrieval of XML Document Fragments Using EXTRIP. In *Proceedings of the Workshop of the Initiative for the Evaluation of XML Retrieval (INEX'2003).*

[28] D. Fallside., P. Walmsley. XML Schema part 0: Primer Second Edition W3C, October 2004, http://www.w3.org/TR/xmlschema-0/

[29] S. Flesca, G. Manco, E. Masciari, L. Pontieri and A. Pugliese. Detecting Structural Similarities Between XML Documents. In *Proceedings of the 5th International ACM SIGMOD Workshop on The Web and Databases (WebDB)*, (2002) pp. 55-60.

[30] N. Fuhr. Probabilistic models in information retrieval. *The Computer Journal 35, 3*, (1992) pp. 243-255.

[31] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval. In: *Proceedings of ACM-SIGIR*, New Orleans, (2001) pp. 172-180.

[32] P. Ganesan, H. Garcia-Molina and J. Windom. Exploiting Hierarchical Domain Structure to Compute Similarity. *ACM Transactions on Information Systems (TOIS)*, *Volume 21, Issue 1*, (2003) pp. 64-93.

[33] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadrt and K. Shim. Xtract: A System for Extracting Document Type Descriptors form XML Documents. In *Proceedings of ACM SIGMOD*, (2000) pp. 165-176.

[34] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the VLDB Conference*, (1997) pp. 436-445.

[35] J. C. Gower and G. J. S. Ross. Minimum Spanning Trees and Single Linkage Cluster Analysis, *Applied Statistics, 18,* (1969) pp. 54-64.

[36] T. Grabs and H.-J. Schek. Generating Vector Spaces On-the-fly for Flexible XML Retrieval. In *Proceedings of ACM SIGIR'02 Workshop on XML and information Retrieval*, (2002) pp.4-13.

[37] S. Guha, H.V. Jagadish, N. Koudas, D. Srivastava and T. Yu. Approximate XML Joins. In *Proceedings of ACM SIGMOD*, (2002) pp. 287-298.

[38] S. Guha, N. Koudas, D. Srivastava and T. Yu. Index-Based Approximate XML Joins. In *Proceedings of the ICDE Conference*, (2003) pp. 708-710.

[39] M. Halkidi, Y.Batistakis and M. Vazirgiannis. Clustering Algorithms and Validity Measures, in *Proceedings of the SSDBM Conference*, USA, (2001) pp. 3-22.

[40] S. Helmer, Measuring the Structural Similarity of Semistructured Documents Using Entropy. In *Proceedings of the VLDB'07 Conference*, (2007), pp. 1022-1032.

[41] D.S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. In *Communications of the ACM*, *18:6,* (1975) pp 341-343.

[42] S. Joshi, N. Agrawal, R. Krishnapuram and S. Negi. A Bag of Paths Model for Measuring Structural Similarity in Web Documents. In *Proceedings of the ACM SIGKKD Conference on Knowledge Discovery and Data Mining*, USA, (2003), pp. 577-582.

[43] A.M. Kade and C.A. Heuser. Matching XML Documents in Highly Dynamic Applications. In *Proceeding of the 8th ACM Symposium on Document Engineering (DocEng'08)*, Brazil, (2008) pp.191-198.

[44] K. Kailing, H.P. Kriegle, S. Schonauer and T. Seidl. Efficient Similarity Search for Hierarchical Data in Large Databases. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT 04)*, Greece, (2004) pp. 676-693.

[45] H.P. Kriegel and S. Schönauer. Similarity Search in Structured Data. In *Proceedings of the 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 03)*, Czech Republic, (2003) pp. 309-319.

[46] J. H. Lee. Properties of extended Boolean models in information retrieval. In *Proceedings of the ACM SIGIR Conference,* Springer-Verlag New York, (1994) pp.182-190.

[47] L. Leitao, P. Calado and M. Weis. Structure-Based Inference of XML Similarity for Fuzzy Duplicate Detection. In *Proceedings of the 16th ACM conference on Conference on Information and Knowledge Management (CIKM'07)*, Portugal, (2007) pp. 293-302.

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

[49] W. Lian, D. Cheung, N. Mamoulis and S.M. Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge and Data Engineering*, *Vol. 16, 1*, (2004) pp. 82-96.

[50] W. Liang and H. Yokota. SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes. *Transactions of Information Processing Society of Japan, Volume 47,* (2006) pp. 47-57.

[51] W. Liang and H. Yokota. LAX: An Efficient Approximate XML Join Based on Clustered Leaf Nodes for XML Data Integration. In *Proceedings of BNCOD'05*, Springer LNCS 3567, (2005) pp 82-97.

[52] D. Lin. An Information-Theoretic Definition of Similarity. In *Proceedings of the 15th International Conference on Machine Learning*, (1998) pp. 296-304.

[53] A. G. Maguitman, F. Menczer, H. Roinestad and A. Vespignani. Algorithmic Detection of Semantic Similarity. In *Proceedings of the 14th International WWW Conference*, Japan, (2005) pp.107-116.

[54] F. Mandreoli, R. Martoglia and P. Tiberio. Approximate Query Answering for a Heterogeneous XML Document Base. In *Proceedings of the 5th WISE (Web Information Systems Engineering) Conference*, (2004) pp. 337-351.

[55] A. Marian, S. Abiteboul and L. Mignet. Change-Centric Management of Versions in an XML Warehouse, In *Proceedings of* the *VLDB Conference*, (2001) pp. 581-590.

[56] A. Marian, S. Amer-Yahia, N. Koudas and D. Srivastava. Adaptive Processing of Top-k Queries in XML. In *Proceedings of the ICDE Conference*, (2005) pp. 162-173.

[57] H. Meuss. *Logical Tree Matching with Complete Answer Aggregates for Retrieving Structured Documents*. PhD thesis, University of Munich, 2000.

[58] G. Miller. WordNet: An On-Line Lexical Database. *Int. Journal of Lexicography, 3,* (1990) pp. 235-244.

[59] E. Myers. An O(ND) Difference Algorithm and Its Variations. *Algorithmica*, *1, 2*, (1986) pp. 251-266.

[60] B. Nguyen, S. Abiteboul, G. Cobena and M. Preda. Monitoring XML Data on the Web. In *Proceedings of ACM SIGMOD*, (2001) pp. 437-448.

[61] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the 5th ACM SIGMOD International Workshop on the Web and Databases (WebDB)*, (2002) pp. 61-66.

[62] J. Pokorny and V. Rejlek. A Matrix model for XML Data. Chap. in: Databases and Information Systems, *Selected Papers from the Sixth International Baltic Conference DB&IS´2004*, *V. 118 Frontiers in Artificial Intelligence and Applications*, Ed. J. Barzdins and A. Caplinskas, IOS Press, (2005) pp. 53-64.

[63] R. Quinlan. Data mining tools see5 and c5.0, 2004..

[64] D. Rafiei, D. Moise and D. Sun. Finding Syntactic Similarities between XML Documents. In *Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA),* (2006) pp. 512-516.

[65] C. J. Rijsbergen Van. *Information Retrieval*, Butterworths, London, 1979.

[66] C. J. Rijsbergen Van. Introduction to Information Retrieval, In *Proceedings of EARIA 06* (*Ecole d'Automne en Recherche d'Information et Application*), *Chapter 3*, 2006.

[67] J. Robie. XQL (XML Query Language). http://metalab.unc.edu/xql/xql-proposal.xml, 1999.

[68] N. Roussopoulos, S. Kelley and F. Vincent. Nearest Neighbor Queries. In *Proceedings of ACM SIGMOD*, ACM Press, (1995) pp. 71-79.

[69] G. Salton. *The SMART Retrieval System*, Prentice Hall, New Jersey, 1971.

[70] G. Salton and M.J. Mcgill. Introduction to Modern Information Retrieval, McGraw-Hill, Tokio, 1983.

[71] G. Salton And C. Buckley. Term-Weighting Approaches in automatic Text Retrieval. *Information Processing and Management: an International Journal*, *24(5),* (1988) pp. 513-523.

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

[73] R. Schenkel, A. Theobald and G. Weikum. Semantic Similarity Search on Semistructured Data with the XXL Search Engine, *Information Retrieval*, *8,* (2005) pp. 521-545.

[74] T. Schlieder. Similarity Search in XML Data Using Cost-based Query Transformations. In *Proceedings of the 4th ACM SIGMOD International Workshop on the Web and Databases (WebDB)*, (2001) pp. 19-24.

[75] T. Schlieder and H. Meuss. Querying and Ranking XML Documents. *Journal of the American Society for Information Science, Spec. Top. XML/IR 53(6)*, (2002) pp. 489-503.

[76] H. Schöning. Tamoni – A DBMS Designed for XML. In *Proceedings of the ICDE Conference*, (2001) pp. 149-154.

[77] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In *Pattern Matching in Strings, Trees and Arrays*, chapter 14, Oxford University Press, 1995.

[78] K.C. Tai. The Tree-to-Tree correction problem. In *Journal of the ACM*, *26*, (1979) pp. 422-433.

[79] J. Tekli, R. Chbeir and K. Yetongnon. Semantic and Structure based XML Similarity: An Integrated Approach. In *Proceedings of the 13th Interventional Conference on Management of Data (COMAD'06)*, New Delhi, India, (2006) pp. 32- 43.

[80] J. Tekli, R. Chbeir and K. Yetongnon. Structural Similarity Evaluation between XML Documents and DTDs. In *Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE'07)*, Springer-Verlag Berlin Heidelberg (LNCS 4831), Nancy, France, (2007) pp. 196-201.

[81] A. Theobald and G. Weikum. Adding Relevance to XML. In *Proceedings of the 3rd International Workshop on the Web Databases (WebDB'00)*, USA, (2000) pp. 105-124.

[82] A. Tversky. Features of Similarity. *Psychological Review, 84(4),* (1977) pp. 327-352.

[83] J. Wagner and M. Fisher. The String-to-String correction problem. *Journal of the Association of Computing Machinery*, *21, 1*, (1974) pp. 168-173.

[84] Y. Wang, D.J. Dewitt and J.Y. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *Proceedings of the ICDE Conference*, (2003) pp. 519-530.

[85] N. Webber, C. O'connel, B. Hunt, R. Levine L., Popkin and G. Larose. *The Information and Content Exchange (ICE) Protocol*. http://www.w3.org/TR/NOTE-ice, 2000.

[86] M. Weis and F. Naumann. Dogmatix Tracks down duplicates in XML. In *Proceedings of the ACM SIGMOD Conference,* USA, (2005) pp. 431-442.

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

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

[89] K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal of Computing, 18(6),* (1989) pp.1245-1262.

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