Lebanese American University Repository (LAUR)

Post-print version/Author Accepted Manuscript

Publication metadata

Title: Approximate XML structure validation based on document–grammar tree similarity

Authors: Joe Tekli, Richard Chbeir, Agma J.M. Traina, Caetano Traina Jr. and Renato Fileto

Journal: Information sciences

DOI/Link: https://doi.org/10.1016/j.ins.2014.09.044

How to cite this post-print from LAUR:

Tekli, J., Chbeir, R., Traina, A. J., Traina Jr, C., & Fileto, R. (2015). Approximate XML structure validation based on document–grammar tree similarity. *Information Sciences*. Doi: 10.1016/j.ins.2014.09.044

© 2015

This Open Access post-print is licensed under a Creative Commons Attribution-Non Commercial-No Derivatives (CC-BY-NC-ND 4.0)

This paper is posted at LAU Repository

For more information, please contact: archives@lau.edu.lb

Approximate XML Structure Validation based on Document-Grammar Tree Similarity

Joe Tekli [1], Richard Chbeir [2 *], Agma J.M. Traina [3], Caetano Traina Jr. [3], Renato Fileto [4]

[1] *Dept. of Elec. and Compt. Eng., SOE, Lebanese American University (LAU), 36 Byblos, Lebanon*
[2] *LIUPPA Laboratory, University of Pau and Adour Countries (UPPA), 64200, Anglet, France*
[3] *ICMC, University of Sao Paulo (USP), 13566-590 - São Carlos, SP, Brazil*
[4] *Federal University of Santa Catarina (UFSC), 88040-900 - Florianopolis, SC, Brazil*

Abstract. Comparing XML documents with XML grammars, also known as XML document and grammar validation, is useful in various applications such as: XML document classification, document transformation, grammar evolution, XML retrieval, and the selective dissemination of information. While exact (Boolean) XML validation has been extensively investigated in the literature, the more general problem of approximate (similarity-based) XML validation, i.e., document-grammar similarity evaluation, has not yet received strong attention. In this paper, we propose an original method for measuring the structural similarity between an XML document and an XML grammar (DTD or XSD), considering their most common operators that designate constraints on the existence, repeatability and alternativeness of XML elements/attributes (e.g., ?, *, *MinOccurs*, *MaxOccurs*, etc.). Our approach exploits the concept of tree edit distance, introducing a novel edit distance recurrence and dedicated algorithms to effectively compare XML documents and grammar structures, modeled as ordered labeled trees. Our method also inherently performs exact validation by imposing a maximum similarity threshold (minimum edit distance) on the returned results. We implemented a prototype and conducted several experiments on large sets of real and synthetic XML documents and grammars. Results underline our approach's effectiveness in classifying similar documents with respect to predefined grammars, accuratly detecting document and/or grammar modifications, and performing document and grammar relevance ranking. Time and space analysis were also conducted.

Keywords: XML; Semi-structured data; XML Grammar; Structural Similarity; Tree Edit Distance; Classification, Relevance Ranking.

1. Introduction

The structural and self-describing nature of XML promotes a number of emerging techniques ranging from XML version control, intelligent Web search, and data integration, to message translation and clustering/classification, requiring, in one way or another, some notion of XML structural similarity. In XML similarity-related research, most work has focused on estimating similarity at the XML data layer (comparing XML documents, e.g., [26, 33, 48]), while quite a few studies have targeted the XML type layer (comparing XML grammars, e.g., [5, 28, 61]). Nonetheless, few efforts have been dedicated to similarity evaluation in-between the XML data and type (document/grammar) layers.

Traditionally, most studies related to XML document/grammar comparison have targeted XML validation [7, 8, 49], i.e., a specific case of Boolean XML comparison, designed to verify whether an XML document is valid (or not) with respect to (w.r.t.) a given XML grammar (DTD [16] or XSD [31]). Yet with the proliferation of heterogeneous XML data on the Web (i.e., documents originating from different data-sources and not conforming to the same grammar, or documents lacking predefined grammars), there is an increasing need to perform ranked XML document/grammar comparison, which we refer to as 'approximate XML validation': identifying those documents which are not necessarily valid w.r.t. the user grammar, but which share a certain amount of similarity with the grammar, ranked following their similarity scores.

Evaluating the similarity between heterogeneous documents and grammars can be exploited in various application scenarios requiring accurate and ranked detection of XML structural similarities [10, 62], ranging over: XML document classification against a set of grammars declared in an XML database [10, 80], (just as DB schemas are necessary in traditional DBMS for the provision of efficient storage, retrieval and indexing facilities, the same is true for DTDs and/or XSDs in XML repositories), XML ranked document retrieval via structural queries [32, 55] (a structural query being represented as a DTD/XSD in which additional constraints on content can be defined), the selective dissemination of XML documents [10] (user profiles being expressed as DTDs/XSDs against which the incoming XML data stream is matched), as well as Web service matching and SOAP processing (searching and ranking services which best match WSDL[1] service requests, and comparing outgoing SOAP messages to sender-side WSDLs, processing only those parts of the messages which differ from the WSDL descriptions in order to avoid unnecessary overhead, and thus reduce processing cost in SOAP parsing [74], serialization [2], and communications [72, 78]).

---

* Corresponding author. Tel.: +33559574337; fax: +33559574308; e-mail: richard.chbeir@univ-pau.fr

[1] Web Service Description Language (WSDL) is a special XML grammar structure that supports the machine-readable description of a Web service's interface and the operation it supports, including corresponding SOAP message formats.

In this study, we focus on the problem of evaluating the structural similarity between an XML document and an XML grammar, i.e., comparing the structural arrangement and ordering of XML elements/attributes in the XML document and the XML grammar. Different from previous approaches which are either generic (disregarding XML grammar constraints, e.g., the *Or* operator, ?, *, +, etc.) [32, 50, 75], developed for the DTD language and do not consider more complex and expressive XSD-based constraints (e.g., *MinOccurs* and *MaxOccurs*) [9, 10], or restricted to Boolean results (i.e., traditional XML validation methods [7, 8, 49]), we aim at providing a method which is:

- Fine-grained in detecting and identifying the structural similarities and disparities between XML documents and grammars, in comparison with current generic [32, 75] and alternative [9, 10] approaches,
- Considering the more expressive XSD grammar constraints (namely *MinOccurs* and *MaxOccurs*), in comparison with less expressive DTD-based constraints (e.g., ?, *, +) handled in existing methods [9, 10],
- Producing a ranked similarity result, in comparison with existing Boolean (validation) methods, e.g., [7, 8, 49].

To achieve these goals, we provide a new approach that extends well-known dynamic programming techniques for finding the edit distance between tree structures, XML documents and grammars being modeled as Rooted Ordered Labeled Trees. Our approach consists of two main phases: i) XML document/grammar tree representation, and ii) XML document/grammar tree comparison (cf. overall architecture in Fig. 1). While XML documents can be naturally represented as labeled trees, XML grammars are usually more intricate, due to the various types of constraints on the existence, repeatability and alternativeness of XML nodes (e.g., *?*, *, + operators in DTDs, *MinOccurs*, *MaxOccurs* cardinality operators in XSD, as well as the *And* sequence operator and *Or* alternativeness operator). These would have to be considered to obtain an accurate similarity measure. Hence, we address the problem of comparing an XML document with an XML grammar as that of: producing a tree representation for the XML grammar (comparable to the XML document tree representation) with additional components to describe cardinality constraints (namely the *MinOccurs* and *MaxOccurs* operators), and then applying a tree-to-tree edit distance function to compute document-to-grammar structural similarly, taking into account XML grammar constraints. We introduce dedicated grammar transformation rules to simplify grammar expressions (while preserving their expressiveness) representing each grammar as a single tree or a set of trees following its *disjunctive normal form* (i.e., a set of grammars free of the *Or* operator, e.g., declaration (*a* | (*b*, *c*)) is split into two declarations: *a* and (*b*, *c*)), each being represented as a separate tree). Then, we introduce a *Tree* (*Edit Distance*) *Comparison* approach to compute (concurrently, using multi-thread processing), the cost of transforming the XML document tree so that it becomes valid w.r.t. the (set of) XML grammar tree(s). Minimum *Tree edit Operations Costs* computed via *TOC* module, are fed to a *Tree Edit Distance* (*TED*) algorithm, which identifies the minimum distance (maximum similarity) value. We build on *TED* as an effective and efficient means to compare semi-structured data, e.g., XML documents [18, 26, 48], which has been proven optimal in structural similarity evaluation, w.r.t. less accurate methods [17]. Also, note that our XML grammar tree model considers complex declarations, including: i) repeatable sequence expressions, ii) repeatable alternative expressions, and iii) recursive expressions, which have been disregarded in most existing studies, e.g., [9, 32, 57]. In addition, our grammar tree model is not limited to context-free (DTD-like) grammar declarations: where the definition of an element is unique and independent of its position in the grammar; but can be used with context-sensitive (XSD-based) declarations: where identically labeled elements can have multiple definitions in different contexts in the grammar.

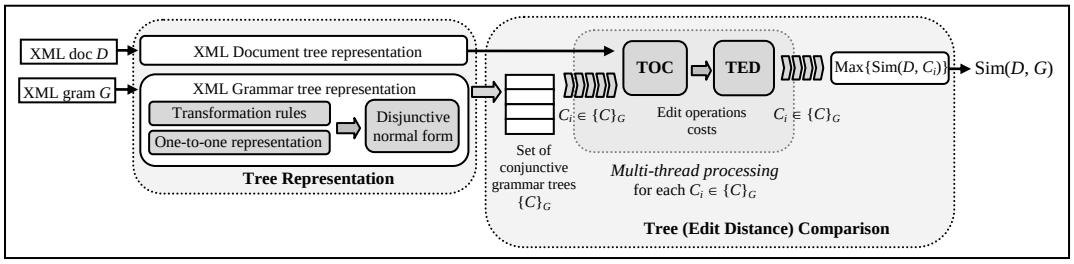


Fig. 1. Simplified activity diagram describing our XML document/grammar comparison framework.

A prototype system called *XS3* (XML Structure & Semantic Similarity) has been developed to evaluate and validate our approach, conducting a large battery of experiments on large XML datasets, covering: *One to One* (comparing one document to one grammar), *One to Many* (comparing one XML document to a set of grammars and vice-versa) and *Set comparison* (enabling XML document/grammar classification and ranked retrieval). Results highlight fine-grained (accurate) similarity scores, produced in typical case polynomial time.

The remainder of the paper is organized as follows. Section 2 presents preliminary notions. Section 3 describes our XML grammar tree representation model. Our XML document-grammar structure comparison algorithms are developed in Section 4. Section 5 presents the experimental tests. Section 6 briefly reviews the state of the art in XML document/grammar similarity approaches and related problems. Section 7 concludes the paper.

2. Preliminaries

2.1. XML Document Representation Model

Following the Document Object Model (DOM) [77], XML documents represent hierarchically structured information and can be represented as *rooted ordered labeled trees*.

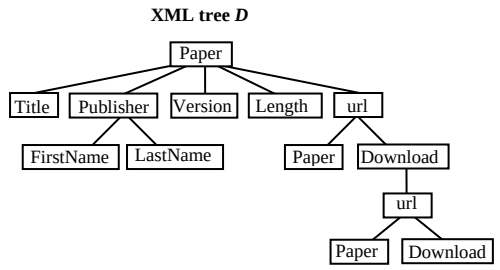
Definition 1 – Rooted Ordered Labeled Tree: It is a rooted tree in which the nodes are labeled and ordered. We denote by *T[i]* the ith node of *T* in preorder traversal, *T[i].ℓ* its label, *T[i].d* its depth, and *T[i].Deg* its out-degree (i.e., the node's fan-out). *R*(*T*)=*T*[0] designates the root node of tree *T*. In the remainder of this paper, terms *tree* and *rooted ordered labeled tree* are used interchangeably ●

Definition 2 – XML Document Tree: It is a *rooted ordered labelled tree* in which the nodes represent XML elements/attributes, labelled following element/attribute tag names. Element nodes are ordered following their order of appearance in the XML document. Attribute nodes appear as children of their encompassing element nodes, sorted left-to-right by attribute name, and appearing before sub-element siblings [48, 83] ●

Note that the order of attributes (unlike elements) is irrelevant in native XML [1], yet in the context of XML structure comparison and processing, attribute nodes are usually ordered (as described above) so as to reduce the complexity of the similarity evaluation process [48, 83]. Element/attribute values can be disregarded (*structure-only*) or considered (*structure-and-content*) in the comparison process following the application scenario (e.g., structure-only comparison is usually performed when processing heterogeneous documents for clustering/classifying [26, 48], whereas data values are generally considered in XML change management and data integration [25, 42]). In this paper, we address heterogeneous XML document-grammar comparison, and thus target element/attribute tag names (*structure-only* comparison) rather than data values. A sample XML document structure is depicted in Fig. 2.a.

```
<?xml?>
<Paper title= "…">
  <Publisher>
      <FirstName>…</FirstName> <LastName>…</LastName>
  </Publisher>
  <Version> … </Version>
  <Length>…</Length>
  <url>
      <Paper>…</Paper>
      <Download>
        <url>
          <Paper>…</Paper> <Download>…</Download>
        </url>
      </Download>
  </url>
</Paper>
```

Sample XML document *Paper.xml* [1]



XML tree representation *D* of *Paper.xml*

a. Sample XML document, and corresponding tree representation.

```
<! DOCTYPE [
<!ELEMENT Paper ((Author* | Publisher), Version, Length?, url?)>
<!ELEMENT Publisher (FirstName?, LastName)>
<!ELEMENT url (Homepage, Download+)>
<!ELEMENT Download (url?)
        <!ELEMENT Author (#PCDATA)>
        <!ELEMENT Version (#PCDATA)>
        <!ELEMENT Length (#PCDATA)>
        <!ELEMENT FirstName (#PCDATA)>
        <!ELEMENT LastName (#PCDATA)>
        <!ELEMENT Homepage (#PCDATA)>  ]
```

XML grammar in DTD syntax

$V_{Paper} \rightarrow (W_{Author*} \mid V_{Publisher}), W_{Version}, W_{Length}?, V_{url}?$

$V_{Publisher} \rightarrow W_{FirstName?}, W_{LastName}$

$V_{url} \rightarrow W_{Homepage}, W_{Download+}$

$V_{Download} \rightarrow V_{url}?$

Production rules (structural model definitions) describing the structure of the DTD grammar above in formal language

```
<?XML?>
<schema>
 <element name= "Paper">
  <sequence> <choice>
    <element name= "Author" minoccurs="0" maxoccurs= "unbounded"/>
    <element name= "Publisher">
      <sequence>
        <element name= "FirstName" minoccurs="0" type= "String"/>
        <element name= "LastName" type= "String"/>
      </sequence> </element> </choice>
    <element name= "Version" type= "Decimal"/>
    <element name= "Length" minoccurs= "0" type= "Decimal"/>
    <element name="url" minoccurs= "0">
      <sequence>
        <element name="Homepage" type= "URI"/>
        <element name= "Download" maxoccurs= "unbounded" type="URI">
          <element ref=  "url" minoccurs = "0"/>
        </element>
      </sequence> </element></sequence> </element>
</schema>
```

XML grammar in simplified XSD syntax (allowing a higher degree of expressiveness in defining structural constraints and data-types)

b. Sample XML grammars, in DTD and XSD syntaxes.

Fig. 2. Sample XML document and XML grammas.

Note that hyper-links in XML documents (e.g., XLinks and IDREFs) and other types of nodes such as *entities*, *comments* and *notations* are usually disregarded in most existing structure comparison methods, e.g., [18, 26, 30, 33, 48], since they are not considered part of the core structure of XML documents.

[1] The dots indicate the place where XML element/attribute data values reside.

2.2. XML Grammar Representation Model

An XML grammar (e.g., DTD [16] or XSD [31]) is an entity consisting of a set of expressions describing XML element/attribute structural positions and data-types, and defining the rules elements/attributes adhere to in corresponding document instances (cf. Fig. 2.b). The structural properties of XML grammars are basically captured by *regular tree languages* [46], XML grammars being viewed as special *regular tree grammars* [21, 46, 47]. In formal language theory [34], a regular tree grammar consists of a set of production rules to transform trees. Formally:

Definition 3 – Regular Tree Grammar: It is represented as a tuple $G = (N, T, R, p)$ where N is a set of non-terminal symbols[1], T is a set of terminal symbols, R is a set of regular expressions over $N \cup T$, and p is a function p: $N \rightarrow R$ that associates a non-terminal symbol $n \in N$ with a regular expression $r_n \in R$, producing a set of production rules of the form $n \rightarrow r_n$. The language $L(G)$, defined based on grammar G, consists of all the possible trees that can be generated following the set of symbols and production rules defined in G [46] ●

Definition 4 – XML Grammar: It can be viewed as a special *regular tree grammar* [21, 46, 47], where each symbol underlines an element *e*, such that non-terminal symbols underline composite XML element labels, terminal symbols underline simple (leaf node) element labels or attribute labels, and where the right hand side of their production rules $e \rightarrow r_e$ are made of special regular expressions r_e which we identify as *structural models* (or *structural expressions*), defined using combinations of *XML grammar constraint operators* (instead of traditional regular expression operators). XML grammar constraint operators specify rules on the existence and repeatability of elements/attributes, namely: *cardinality constraints*, i.e., *?*, *, + in DTDs, *MinOccurs* and *MaxOccurs* in XSDs, and *alternativeness constraints*: *And* (sequence) and *Or* (choice) operators. In addition, special production rules are introduced in XML grammar languages (which do not exist in traditional tree languages [34]) to encode XML element data-type content models (e.g., *#PCDATA, String, Decimal, gYear,* cf. Fig. 2.b) ●

Note that the DTD language [16] allows *context-free-grammars* (*local tree grammars*) [46], which means that the structural model associated to an given element is independent of its position (i.e., context) in the document, the element being identified by its label (i.e., for an element *e* in grammar *G*, there exists only one possible production rule $e \rightarrow r_e$, i.e., only one possible structural model r_e). In contrast, XSD [31] allows *context-sensitive grammars* (*single type tree grammars*) [34] where the structural model associated to an element depends on its position in the document (e.g., one might have more than one production rule sharing the same element *e* in the grammar, e.g., $e \rightarrow r_e$ and $e \rightarrow r_e$', following the element's structural position). For further details, a study highlighting the correlation between XML grammar languages (namely DTD and XSD) and regular tree languages can be found in [46].

2.3. XML Document/Grammar Structural Similarity

We identify two kinds of XML document/grammar structure similarity: i) Boolean comparison, referring to traditional *XML structure validation*, and ii) ranked comparison, which we refer to as *approximate XML structure validation*.

Definition 5 – XML Structure Validation (Boolean Comparison): denoted $G \vDash D$, an XML document (tree) D is deemed valid w.r.t. an XML (regular tree) grammar G (i.e., D conforms to G), if all element (attribute) tags in D match the element (attribute) structural models defined in G, considering structural model constraint operators. In other words, the result of the validation operation would be a Boolean value (true or false) indicating whether the document is valid (or not) w.r.t. the grammar, which comes down to checking whether the document tree is included in the language defined by the grammar, i.e., if $D \in L(G)$ ●

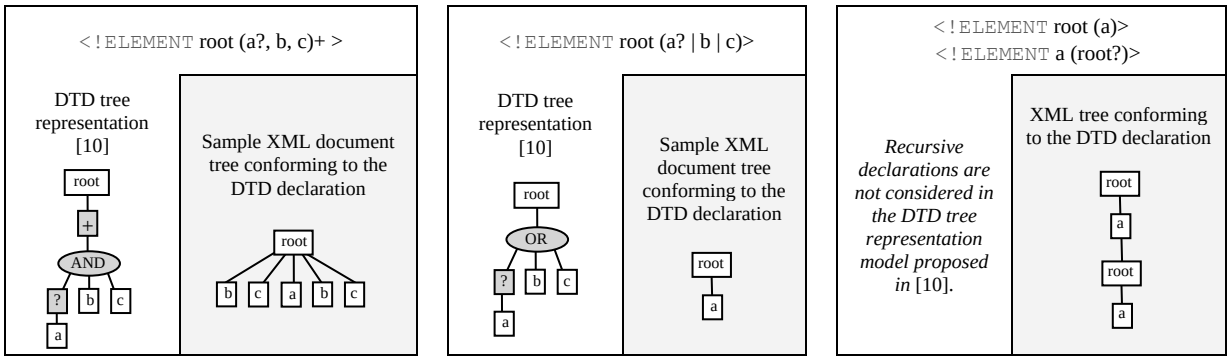
Definition 6 - Approximate XML Structure Validation (Ranked Comparison): denoted $G \mid\approx_{\sigma \in [0,1]} D$, approximate XML structure validation between an XML document (tree) D and an XML (regular tree) grammar G, with a similarity (relevance) score $\sigma \in [0, 1]$ (i.e., D approximately conforms to G with a similarity score σ), is defined as the structural comparison (matching) between the element/attribute tag names in D and the structure models in G, in order to determine the best matches possible. Corresponding (best) matching scores are compiled into an overall similarity (relevance) score σ, highlighting the structural relatedness between D and G. In other words, similarity score σ underlines the degree of membership of D w.r.t. the grammar (regular tree) language $L(G)$ ●

Note that in the remainder of the paper, we sometimes use the simple notation: $G \mid\approx D$ to designate that document D approximately validates grammar G (omitting similarity score σ for ease of presentation). Also note that we adopt the concept of *similarity* as the inverse of a *distance function*, i.e., a smaller distance value underlining a higher similarity between the XML document and grammar being compared, and vice-versa.

[1] In language theory, terminal symbols are those not assigned to production rules, and thus cannot be broken down to smaller units.

3. XML Grammar Tree Representation

The main idea is to compute a tree representation of the XML grammar in order to apply a tree-to-tree edit distance for computing the document/grammar similarity. To do so, we aim to unfold the XML (regular tree) grammar *G* structural expressions into a set of conjunctive grammar trees {*C*} of equivalent expressiveness, such that comparing a document tree *D* with grammar *G* would come down to comparing *D* with {*C*}. Here, the main difficulties in XML document/grammar tree-to-tree comparison lie within the disparities in the representation and processing of: i) repeatable expressions defined via the *And* operator (cf. Fig. 3.a), ii) alternative declarations defined via the *Or* operator (cf. Fig. 3.b), and iii) recursive declarations (which could induce infinite loops of elements, cf. Fig. 3.c).



a. Repeatable expression **b.** Alternative declaration **c.** Recursive declaration

Fig. 3. Disparities in tree representations between XML document and grammar structures, following the grammar (DTD) tree representation model in [10] (one of the central methods in the literature).

Intuitively, the higher the disparities in document and grammar tree representations, the more complicated it becomes to perform the tree comparison (matching) task. Hence, we need to have expressive, yet simplified (flattened) XML grammar trees, which are (more easily) comparable to XML document trees. To do so, we proceed in three phases:

- ***XML Grammar Transformation Rules*:** First, we introduce a number of transformation rules to flatten XML grammar declarations, considering the most common XML grammar constraints.
- ***One-to-One Document/Grammar Representation*:** Second, we extend transformation rules to further simplify repeatable and recursive declarations in the grammar w.r.t. each document tree being compared (one-to-one).
- ***XML Grammar Tree Model*** based on the ***Disjunctive Normal Form*:** Where the resulting simplified (flattened) grammar is represented as a set of conjunctive grammars made solely of *sequence* declarations (i.e., elements connected via an *And* operator), eliminating *alternative* declarations (i.e., elements connected via the *Or* operator), producing grammar tree structures which are (more easily) comparable to document trees.

The remainder of this section develops each of the phases mentioned above, and provides examples.

3.1. XML Grammar Transformation Rules and Properties

An XML grammar transformation rule can be viewed as a binary function that transforms an XML structural expression into another, thus transforming one grammar into another. Formally:

Definition 7 – XML Grammar Transformation: Let Ω denote the domain of XML grammar structural expressions (set of all grammar structural expressions allowed in our study, cf. Definition 4), a transformation rule R is defined as a function R: $\Omega \rightarrow \Omega$, associating an input structural expression $r_e \in \Omega$ with an output structural expression $r_e' \in \Omega$, such that r_e' results from the application of transformation rule R on r_e, denoted $r_e \rightarrow^R r_e'$. When applied to all the structural expressions in an XML grammar G, i.e., $\forall\ r_e \in G$, $r_e \rightarrow^R r_e'$, we say that R is applied to G, and transforms it into an output grammar G' made of the transformed expressions $\forall\ r_e' \in G'$, denoted $G \rightarrow^R G'$ ●

Definition 8 – *Information Structure Preserving* (*ISP*) property: Given an XML grammar (structural expression in) G and a grammar transformation rule R applied to G, resulting in G', i.e., $G \rightarrow^R G'$, rule R is deemed *information structure preserving* if any XML document tree D that conforms to G also conforms to G' and vice-versa, i.e., $\forall\ D$, $G \vDash D \Leftrightarrow G' \vDash D$. In other words, the original and the transformed grammar (structural expressions in) G' have the same structural expressiveness, denoted $G \triangleq G'$ ●

The transformation rules we provide in our study (cf. Table 1) verify the *ISP* property in most practical cases (with one exception discussed subsequently), i.e., they maintain the expressiveness of the input grammar's structural models. They can be grouped in three main categories: *simple expression flattening* (Rule 1), *repeatable sequence expression flattening* (Rule 2) and *repeatable alternative expression flattening* (Rule 3).

Hereunder, we utilize a DTD-like syntax (even when presenting XSD operators) to ease the presentation. We introduce a simplified notation for *MinOccurs* and *MaxOccurs*, such that an element (expression) *e* that is associated

cardinality constraints: *MinOccurs* = *x* ∧ *MaxOccurs* = *y*, is noted e_x^y. Note that an element (expression) *e* with no associated cardinality constraints is identified as having a *null* constraint, which is equivalent to e_1^1, i.e., it appears exactly once in the XML document. We also highlight the notion of *empty structural model* (utilized in defining our transformation rules): given an XML grammar *G*, an element $e \in G$ has an *empty structural model*, noted $e \to \perp$, (i.e., $r_e \equiv \perp$) when *e* does not encompass any sub-elements, and corresponds to a leaf node in the XML document tree instance.
Recall that we only target XML structure in our current study, and hence do not discuss element content data-types and values. Thus, elements with basic content models (e.g., *PCDATA*, *String*, *Integer*, etc.) will be processed as empty structural models (e.g., <!ELEMENT *dummy* (*#PCDATA*)>, will be processed as production rule: *dummy* → ⊥).

Table 1. Outline of our XML grammar transformation rules.
Note that *A* and *B* designate XML grammar structural expressions.

N#	Rule	Type	*ISP* property
1	$(A_x^y)_u^v \to^{R1} A_{x\times u}^{y\times v}$ (general rule, handling both *MinOccurs* and *MaxOccurs*[1])	Simple expression flattening	Special case
2.1	Simplified version of Rule 2 handling the *MinOccurs* constraint: $(A, B)_x \to^{R2.1} (A , B) , \ldots, (A , B)$ where (A , B) is repeated ***x*** times.	Repeatable sequence expression flattening (And)	✓
2.2	Simplified version of Rule 2 handling the *MaxOccurs* constraint: $(A, B)^y \to^{R2.2} ((A , B)\mid \perp) , \ldots, ((A , B)\mid \perp)$ where $((A , B)\mid \perp)$ is repeated ***y*** times.		
2	$(A, B)_x^y \to^{R2} (A , B) , \ldots, (A , B), ((A , B)\mid \perp) , \ldots, ((A , B)\mid \perp)$ where (A , B) is repeated ***x*** times, and $((A , B)\mid \perp)$ is repeated $\mathbf{z = y - x}$ times		
3.1	Simplified version of Rule 3 handling the *MinOccurs* constraint: $(A \mid B)_x \to^{R3.1} (A \mid B) , \ldots, (A \mid B)$ where $(A \mid B)$ is repeated ***x*** times.	Repeatable alternative expression flattening (Or)	✓
3.2	Simplified version of Rule 3 handling the *MaxOccurs* constraint: $(A \mid B)^y \to^{R3.2} (A_0^1 \mid B_0^1) , \ldots, (A_0^1 \mid B_0^1)$ where $(A_0^1 \mid B_0^1)$ is repeated ***y*** times. Note that $(A_0^1 \mid B_0^1)$ underlines that either *A* or *B* can occur, or nothing at all, which is different from $((A , B) \mid \varepsilon)$ used in *Rule 4.β* underlining that *A* and *B* must occur together, or nothing at all.		
3	$(A \mid B)_x^y \to^{R3} (A \mid B) , \ldots, (A \mid B) , (A_0^1 \mid B_0^1) , \ldots, (A_0^1 \mid B_0^1)$ where $(A \mid B)$ is repeated ***x*** number of times, and $(A_0^1 \mid B_0^1)$ is repeated $\mathbf{z = y - x}$ times.		

The transformation rules in Table 1 verify the *ISP* property (cf. proofs in [73]), to the exception of *Rule 1*, which verifies the *ISP* property in some practical cases, but not in the general case:

Lemma 1 – Given an XML grammar expression of the form $(A_x^y)_u^v$, transformation *Rule 1* complies with the *ISP* property when any of the following conditions holds:

- Condition 1: ($x = y = 1$) or ($u = v = 1$) (i.e., $(A_1^1)_u^v \equiv A_u^v$; $(A_x^y)_1^1 \equiv A_x^y$).
- Condition 2: ($x = u = 0$) and ($y = 1$ or $v = 1$) (i.e., $(A_0^1)_0^v \equiv A_0^v$; $(A_0^y)_0^1 \equiv A_0^y$).
- Condition 3: (x = y) and (u = v) (i.e., $(A_x^x)_u^u \equiv A_{x\times u}^{x\times u} \equiv (A_y^y)_v^v \equiv A_{x\times u}^{x\times u} \equiv A_{y\times v}^{y\times v}$).

Rule 1 may not comply with the *ISP* property otherwise ●

For instance, *ISP* holds when transforming DTD expressions such as:
(*A**)*?*, which is equivalent to $(A_0^\infty)_0^1 \to^{R3} A_0^\infty$;
(*A*+)*?* which is equivalent to $(A_1^\infty)_0^1 \to^{R3} A_0^\infty$;
(*A*+)* which is equivalent to $(A_1^\infty)_0^\infty \to^{R3} A_0^\infty$, since Condition 2 of Lemma 1 holds.

[1] Note that the special case of *MaxOccurs = "unbounded"* is covered in the following section.

Likewise, transforming expression $(A_2^2)_3^3 \rightarrow^{R3} A_6^6$, *ISP* holds since Condition 3 of Lemma 1 holds.

However, consider an expression of the form $(A_2^2)_1^2 \rightarrow^{R3} A_2^4$. Here, the *ISP* property does not hold since the resulting expression does not preserve the structural expressiveness of its original counterpart since (neither of Lemma 1's conditions holds): the transformed expression underlines that expression *A* can occur a minimum of *2* times and a maximum of *4* times (i.e., it accepts *2-to-4* occurrences of element *dummy*), whereas the original expression underlines that expression *A* can occur either *2* times, or *4* times only.

In addition, we extend the repeatable (sequence/alternative) expression flattening rules in Table 1 to handle the special cases of *MinOccurs = 'Unbounded'* (infinite repetitions) and recursive declarations, as shown in the following.

3.2. *One-to-One* Document/Grammar Representation

3.2.1. Handling Repeatable Expressions

As described in Table 1, the simplification of repeatable *sequence* and *alternative* expressions, of the form $(A, B)_x^\infty$ and $(A \mid B)_x^\infty$, following *Rule 2* and *Rule 3*, requires the infinite repetition of flattened expressions $((A , B)| \perp)$ and $(A_0^1 \mid B_0^1)$ respectively, in order to verify the *ISP* property. Yet since our method is *one-to-one* in comparing one single XML document to one single XML grammar, repeatable sequence/alternative expressions can be further simplified without loss of expressiveness in the context of the XML document at hand. This can be done by repeating the flattened expressions: i.e., $((A, B)| \perp)$ w.r.t. *Rule 2*, and $(A_0^1 \mid B_0^1)$ w.r.t. *Rule 3*, only a finite number of times in the transformed grammar, necessary to cover all possible structural configurations of the concerned XML elements/expressions following the original grammar expression. Hence, we propose extensions of transformation *Rule 2* (namely *Rule 2.2*) and *Rule 3* (namely *Rule 3.2*) in Table 2.

We note by $(G \triangleq G')_D$ a grammar *G* and a transformed grammar *G'* having the same structural expressiveness w.r.t. a particular document tree *D*. We say that the rule transforming *G* into *G'* verifies the *ISP* property w.r.t. document tree *D*.

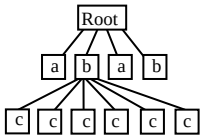
Table 2. Outline of *one-to-one* document/grammar transformation rules.

Note that *A* and *B* designate XML grammar structural expressions

N#	**Rule** (given an XML document tree *D*)	***Type***	***ISP* property**
2.2+	Simplified version of Rule 2+ handling the *MaxOccurs* constraint (*MinOccurs* is handled the same as in *Rule 2.1*): $(A, B)^\infty \rightarrow^{R2.2+} ((A , B)| \perp) , \ldots, ((A , B)| \perp)$ where $((A , B)| \perp)$ is repeated $ceil(\frac{MaxDeg(D)}{\lvert E\rvert})$ times[1], such as E = $(A, B)^\infty$ and $\lvert E\rvert$ denotes the expression's cardinality w.r.t. the main *And* sequence operator (e.g., $\lvert E\rvert$=2 for *E*=(*A*, *B*)*, $\lvert E\rvert$=3 for *E*=(*A*, *B*, *C*)*).	Repeatable *sequence* expression flattening	✓
2+	$(A, B)_x^\infty \rightarrow^{R2+} (A , B) , \ldots, (A , B) , ((A , B)| \perp) , \ldots, ((A , B)| \perp)$ where (*A* , *B*) is repeated *x* times, whereas $((A , B)| \perp)$ is repeated $z = ceil(\frac{MaxDeg(D)}{\lvert E\rvert}) - x$ times[1]		
3.2+	Simplified version of Rule 3+ handling the *MaxOccurs* constraint (*MinOccurs* is handled the same as in *Rule 3.1*): $(A \mid B)^\infty \rightarrow^{R3.2+} (A_0^1 \mid B_0^1) , \ldots, (A_0^1 \mid B_0^1)$ where $(A_0^1 \mid B_0^1)$ is repeated *MaxDeg*(*D*) times.	Repeatable *alternative* expression flattening	✓
3+	$(A, B)_x^\infty \rightarrow^{R3+} (A \mid B) , \ldots, (A \mid B) , (A_0^1 \mid B_0^1) , \ldots, (A_0^1 \mid B_0^1)$ where *(A | B)* is repeated *x* times, and $(A_0^1 \mid B_0^1)$ is repeated $z = MaxDeg(D) - x$ times.		
4	A *strong-linear recursive* expression defined on element *e*, denoted $e_{ref} \Rightarrow e_{rec} \rightarrow^{R4} e_1 \Rightarrow e_2 \Rightarrow \ldots e_{n-1} \Rightarrow e_n$ where e_i denotes i*th* nested occurrence of element *e*, repeated $n = ceil(\frac{Depth(D)+1}{NestDepth(e_{ref}, e_{rec})_G})$ times[1].	Recursive expression flattening	✓

Given an XML grammar (structural expression in) *G*, and an XML document tree *D* to be compared with *G*, the application of *Rule 2+* and/or *Rule 3+* to grammar (structural expressions in) *G*, considering document tree *D*, verifies the *ISP* property w.r.t. *D* (cf. proofs in [73]). More importantly, *Rule 2+* and *Rule 3+* have allowed to simplify (possibly) *infinite* size expressions (originally required to preserve grammar expressiveness in the general case) into expressions which sizes vary linearly w.r.t. the out-degree of the document tree, *MaxDeg*(*D*) (cf. example in Fig. 4.a).

[1] The function *ceil*(*x*) returns the smallest integer value that is not less than *x*. The formulas' proofs are provided in [73].



Sample XML tree *D*

<!ELEMENT Root(a, b)*>
<!ELEMENT b(c*)>

$\rightarrow^{R2+}$

<ELEMENT Root (((a, b)| ⊥), ((a, b)| ⊥), ((a, b)| ⊥))>
<!ELEMENT b(c*)>

Grammar *G*

Grammar *G'* transformed following *Rule 2+*, w.r.t. XML doc tree *D*, such that expression ((*a, b*)| ⊥) is repeated *ceil*($\frac{MaxDeg(D)}{2}$) = 3 times

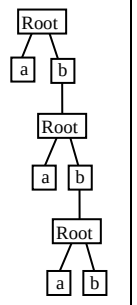
Simplifying expression (*a, b*)* while preserving expressiveness w.r.t. XML document tree *D*
Note that *c** need not be simplified and will be handled by the edit distance algorithm.

a. Flattening repeatable *sequence* expressions via the application of *Rule 2+*.

(1) < ! ELEMENT ***Paper*** (Title , ***Paper*** , (Download | ***Paper****))>
Multiple occurrences of recursive declarations in structural model
Non-linear recursive declaration

(2) < ! ELEMENT ***Paper*** (Title, ***Paper***, (download | url))>
(3) < ! ELEMENT ***Paper*** (Title, (download | ***Paper*?**))>
Strong-linear recursive declarations

b. Sample recursive XML grammar declarations (represented in DTD syntax for ease of presentation).



XML Tree *D*

<ELEMENT Root (a, b)> // *Root*$_{ref}$
<ELEMENT b (Root?)> // *Root*$_{rec}$
<ELEMENT a (#PCDATA)>

≡

<Element name = 'Root'> // *Root*$_{ref}$
<*Sequence*>
<Element name = 'a' type= '#PCDATA'/>
<Element name = 'b'>
<Element ref = 'Root' MinOccurs= '0'/> // *Root*$_{rec}$
</Element>
</*Sequence*>
</Element>

Recursive grammar *G*
(in both DTD and XSD syntaxes)

$\rightarrow^{R4}$

<Element name = 'Root'> // *1st occurrence*
<*Sequence*>
<Element name = 'a' type= '#PCDATA'/>
<Element name = 'b'>
<Element name = 'Root' MinOccurs= '0'> // *2nd occurrence*
<*Sequence*>
<Element name = 'a' type= '#PCDATA'/>
<Element name = 'b'>
<Element name = 'Root' MinOccurs= '0'> *//3rd occurrence*
<*Sequence*>
<Element name = 'a' type= '#PCDATA'/>
<Element name = 'b' type= '#PCDATA'/>
</*Sequence*>
</Element>
</*Sequence*> </Element> </*Sequence*>
</Element>

Non-recursive grammar *G'* flattened while preserving grammar expressiveness w.r.t. document tree *D*
(cannot use DTD syntax here since grammar is context-sensitive)

c. Flattening recursive XML declarations.

Fig. 4. Flattening repeatable and recursive XML declarations.

3.2.2. Handling Recursive Declarations

The problem of handling recursive declarations is comparable to that of handling repeatable expressions such that the recursive expression would have to be repeated (in certain cases) an infinite number of times in the flattened XML grammar to preserve structural expressiveness. Yet, since our method is *one-to-one* in comparing one XML document to one XML grammar, recursive grammar expressions can be simplified without loss of expressiveness in the context of the XML document at hand. To do so, we propose to repeat the recursive nesting only a finite number of times, necessary to cover all possible structural configurations of the concerned XML elements (following the original recursive declaration) in the XML document tree being compared.

Note that in our current study, we only consider *strong-linear recursive* declarations, which are the most common in practice [11, 23], and which are inherently easier to process than *non-linear recursive* expressions. Formally:

Definition 9 – Recursive XML Grammar: An XML grammar *G* is recursive if it contains an element *e* reachable from itself, denoted by $e_{ref} \Rightarrow e_{rec}$, where e_{ref} underlines the original element declaration and e_{rec} its recursive reference in the grammar (e.g., in DTD expression <!ELEMENT *dummy* (*a*, *b*, *dummy*)>, the first dummy element is denoted $dummy_{ref}$ whereas the second is denoted $dummy_{rec}$) ●

Definition 10 – Strong-Linear Recursive XML Grammar: An XML grammar *G* is *strong-linear recursive* if it is *recursive*, and if for each recursive element $e_{ref} \Rightarrow e_{rec}$ in *G*, e_{rec} occurs at most once in the *structural expression* of its containing element, such as e_{rec} is not repeatable, and where every other element reachable from e_{ref} ($\forall e' \in G$ such that $e_{ref} \Rightarrow e'$) is non-recursive [11, 23] (cf. Fig. 4.b) ●

Following *Rule 4*, a *strong-linear recursive* declaration $e_{ref} \Rightarrow e_{rec}$ is transformed into a chain of non-recursive nestings consisting of the elements comprised within e_{Ref} and e_{Rec}, repeated a finite number of times linear in the XML document tree depth (*Depth*(*D*)) and the nesting depth of the recursive declaration (*NestDepth*(e_{Ref}, e_{Rec}) = $e_{Ref}.d$ – $e_{Rec}.d$), in order to preserve structural expressiveness w.r.t. the XML document tree. Hence, given an grammar *G* and an document tree *D* to be compared with *G*, the flattening of *strong-linear recursive* declarations in *G*, following *Rule 4*, produces a transformed grammar *G'* which verifies the *ISP* property w.r.t. *D*, $(G \triangleq G')_D$ (cf. proof in [73]).

A simple example is presented in Fig. 4.c. Here, the recursive declaration defined on node *Root* in grammar *G* is transformed into a chain of non-recursive nestings consisting of the elements comprised within $Root_{Ref}$ and $Root_{Rec}$, repeated ($\frac{5+1}{2}=$) 3 times w.r.t. XML document tree depth (=5) and the nesting depth of the recursive declaration (=2).

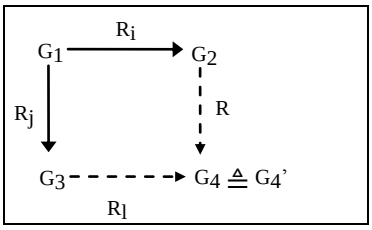
3.3. Applying Transformation Rules

As discussed above, most of our transformation rules verify the *ISP* property to the exception of *Rule 1* which only conditionally complies with *ISP*. As a result, the transformation rules (to the exception of *Rule 1*) verify the *Church-Rosser* property [35, 76] and can be applied to an input XML grammar, in any sequence order, producing an output grammar having the same structural expressiveness (as the input grammar):

Definition 13 – *Extended Church-Rosser* (*ECR*) property: Let Ω be the domain of (structural expressions in) XML grammars, and $\rho = \{R_i, R_j, R_k \ldots\}$ be the set of XML grammar (expression) transformation rules defined on Ω, ρ has the *extended Church-Rosser* property w.r.t. Ω if $\forall\ G_1, G_2, G_3 \in \Omega$, and $\forall\ R_i, R_j \in \rho$, $[\ (G_1 \rightarrow^{Ri} G_2) \wedge (G_1 \rightarrow^{Rj} G_3)\ ] \Rightarrow \exists\ G_4, G_4' \in \Omega, \exists\ R_k, R_l \in \rho$ such that $[\ (G_2 \rightarrow^{Rk} G_4) \wedge (G_3 \rightarrow^{Rl} G_4')$ where $(G_4 \triangleq G_4')\ ]$ (the resulting (structural expressions in) grammars G_4 and G_4' have the same structural expressiveness (cf. *ECR* diagram in Fig. 5.a) ●

In other words, given an input XML grammar *G*, the transformation rules in Table 1 (except the conditional case of *Rule 1*), can be applied to *G* in any sequence order, always resulting in a transformed grammar *G'* having the same structural expressiveness as its original counterpart ($G \triangleq G'$). The same carries for our *one-to-one* document/grammar transformation rules in Table 2 (cf. proof in [73]).

Consider the example in Fig. 5.b. Input grammar declaration *root* (*a*, (*b* | (*c*, *d*, *e*)+ | *f*)?), is transformed, without any loss of expressiveness, to *root* (*a*, $(b^1_0$ | (((*c*, *d*, *e*) | ε), ((*c*, *d*, *e*) | ε), ...) | f^1_0), via the application of two different sequences of transformation rules. In the resulting grammar declaration, all repeatable expressions have been flattened, cardinality constraints being uniquely associated to single elements (i.e., b^1_0 and f^1_0) without loss of expressiveness.



a. Visual description of the *ECR* (*Extended Church-Rosser*) property, w.r.t. XML grammar transformation.

Input grammar declaration *G* : <!ELEMENT *root* (*a*, (*b* | (*c*, *d*, *e*)+ | *f*)?) >	
$G \rightarrow^{R3} G_1 \rightarrow^{R1} G_2 \rightarrow^{R2} G_3$	$G \rightarrow^{R2} G_1' \rightarrow^{R3} G_2' \rightarrow^{R2} G_3'$
Start: *root* (*a*, (*b* | (*c*, *d*, *e*)$^{\infty}_1$ | *f*)$^1_0$)	***Start*** : *root* (*a*, (*b* | (*c*, *d*, *e*)$^{\infty}_1$ | *f*)$^1_0$)
Rule 3 : *root* (*a*, **(***b^1_0 **| ((*c*, *d*, *e*)$^{\infty}_1$)$^1_0$ | *f*$^1_0$)**)	***Rule 2*** : *root* (*a*, (*b* | **((*c*, *d*, *e*), (*c*, *d*, *e*)| ε , (*c*, *d*, *e*)| ε , …)** | *f*)$^1_0$)
Rule 1 : *root* (*a*, (*b*$^1_0$ | **(*c*, *d*, *e*)**$^{\infty}_0$ | *f*$^1_0$))	***Rule 3*** : *root* (*a*, **(***b^1_0 **| (** (*c*, *d*, *e*) , (c, d, e)| ε , (*c*, *d*, *e*)| ε , ...)$^1_0$ **|** ***f*** $^1_0$ **)**)
Rule 2 : *root* (*a*, (*b*$^1_0$ | **(((*c*, *d*, *e*) | ε) , ((*c*, *d*, *e*) | ε) , ...)** | *f*$^1_0$))	***Rule 2*** : *root* (*a*, (*b*$^1_0$ | (**((*c*, *d*, *e*) | ε) , ((*c*, *d*, *e*) | ε) , ...)** | *f*$^1_0$))
a. Sample transformation sequence, yielding $G_3 \triangleq G$.	**b.** Sample transformation sequence, yielding $G_3' \triangleq G$.
$G_3 \triangleq G_3'$: <!ELEMENT *root* (*a*, (*b*$^1_0$ | (((*c*, *d*, *e*) | ε) , ((*c*, *d*, *e*) | ε) , ...) | *f*$^1_0$)) >	

b. XML grammar transformation example, using two different sequences of XML grammar transformation rules.

Fig. 5. Visual description and sample application of sample XML grammar transformations preserving the *ECR* property.

Note that the *ISP* and *ECR* properties reflect the *correctness* and *completeness* or the transformation rules. *Minimality* also seems intuitive since the transformation rules have been specifically defined to deal with exactly each of the common grammar constraints considered in our study, and cannot be further reduced. Nonetheless, the nature of the transformation rules applied, during a grammar simplification task, might differ depending on the nature of the grammar expressions (as shown in the example of Fig. 5.b). Hence, despite yielding the same end result (the same transformed grammar), the *minimality* of the number of transformations applied might not always be guaranteed.

3.4. XML Grammar Tree Model based on the *Disjunctive Normal Form*

To produce simple grammar tree structures comparable to document trees, we propose to unfold an XML grammar into a single tree or a set of trees, depending on the occurrences of the *And* (sequence) and *Or* (choice) operators. To do so, we introduce the *disjunctive normal form* of an XML grammar as a set of *conjunctive grammars*:

Definition 14 – Conjunctive XML Grammar: A grammar G is conjunctive if all structural expressions in G are *sequence* expressions, i.e., $\forall\ e \rightarrow r_e$ in G, r_e is made of elements/expressions connected via the *And* operator ●

Definition 15 – XML Grammar Disjunctive Normal Form (*DNF*): The disjunctive normal form (*DNF*) of an XML grammar G is the set of conjunctive grammars, $DNF(G) = \{C\}_G$ which are equivalent in their expressiveness to G taking into account the *alternative* structural expressions in G, i.e., $\forall\ r_e \in G$ such that $e \rightarrow r_e$, where r_e is made of elements/expressions connected via the *Or* operator (cf. Fig. 6) ●

The *disjunctive normal form* of an XML grammar verifies by definition the *ISP* property, such that the grammar's expressiveness (language) is distributed among its constituent conjunctive grammars, $L(G) = \bigcup_{C_i \in \{C\}_G} L(C_i)$, denoted as $G \triangleq DNF(G)$. In other words, given an XML grammar G, and its representation in disjunctive normal form $DNF(G) = \{C\}_G$, any XML document tree D that conforms to G, will conform to at least one of its conjunctive grammars, i.e., $\forall\ G \vDash D, \exists\ C \in DNF(G)$ such that $C \vDash D$ (the proof carries directly from the definition of *DNF).*

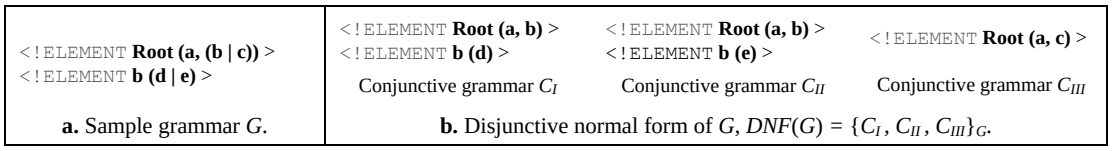


Fig. 6. Representing an XML grammar in its disjunctive normal form.

Note that the number of conjunctive grammars, resulting from the *DNF* of an input XML grammar, depends on the number and configurations of *Or* operators in the input grammar expressions. This may generate a proliferation of conjunctive grammars depending on the expressiveness of the grammar declarations. In this context, we have conducted a mathematical analysis covering some of the most common configurations of *alternative* (*Or*) expressions, based on surveys of real DTDs and XSDs in [11, 23, 38] (some statistics are provided in Section 5.6). Results show that the most common *alternative* expressions generate a number of conjunctive grammars linear in the number of *Or* operators involved (e.g., in Fig. 6, $|DNF(G)| = 1 +$ number of *Or* operators in $G = 3$), while only certain specific cases (of usually *mixed*: *And*-*Or* expressions) yield polynomial and/or exponential sized *DNF* representations (mathematical details are provided in Appendix I).

As a result, we model each resulting conjunctive grammar $C \in DNF(G)$ as a special *rooted ordered labeled tree*:

Definition 16 – Conjunctive XML Grammar Tree: It is a *rooted ordered labeled tree* in which nodes represent XML element/attributes, labeled following element/attribute tag names, and such that each node is assigned the corresponding element/attribute (*MinOccurs*/*MaxOccurs*) cardinality operator. Element nodes are ordered following their order of appearance in the XML grammar declarations. Attribute nodes appear as children of their encompassing element nodes, sorted left-to-right by attribute name, and appearing before all sub-element siblings (similarly to XML document trees, cf. Definition 2). Formally, We model a conjunctive XML grammar tree as $C = (N_C, E_C, L_C, CC_C, g_C)$:

- N_C is the set of nodes (i.e., vertices) in C
- $E_C \subseteq N_C \times N_C$ is the set of edges underlining the XML element/attribute containment relation
- L_C is the set of labels corresponding to the nodes of C
- CC_C is the set of cardinality constraints associated to the nodes of C
- g_C is a function $g_C : N_C \rightarrow L_C \times CC_C$ that associates a label $l \in L_C$ and a cardinality constraint $cc \in CC_C$ to each node $n \in N_C$, following element/attribute ordering as described above.

We denote by $C[i]$ the ith node of C in preorder traversal, represented as a doublet (l, cc) where $l \in L_C$, and $cc \in CC_C$ are respectively its label and cardinality constraint, referred to as $C[i].l$ and $C[i].cc$ (since cardinality constraints amount to *'MinOccurs'* and *'MaxOccurs'*, we refer to the latter as $C[i].MinOccurs$ and $C[i].MaxOccurs$, cf. Fig. 7.a). $C[i].d$ represents the node's depth in the tree, and $C[i].Deg$ its out-degree. $R(C)=C[0]$ designates the root of tree C ●

Recall that while the order of attribute children is irrelevant in XML [1], yet we represent the latter as ordered tree nodes in both our document and grammar tree models (as described above) in order to reduce the complexity of the similarity computation process[1], while seamlessly affecting the accuracy of the similarity results (since attribute nodes, in both XML document and grammar trees, are ordered in the same way).

The algorithm for transforming an XML grammar into its tree representation in provided in Appendix II.

[1] E.g., most unordered tree distance algorithms are of exponential time, compared to average polynomial ordered tree methods [12].

3.5. Running Example: Sample XML Grammar and Corresponding Tree Representation

Consider XSD grammar *Paper.xsd* in Fig. 7.b, to be compared with XML document tree *D* in Fig. 2.a.

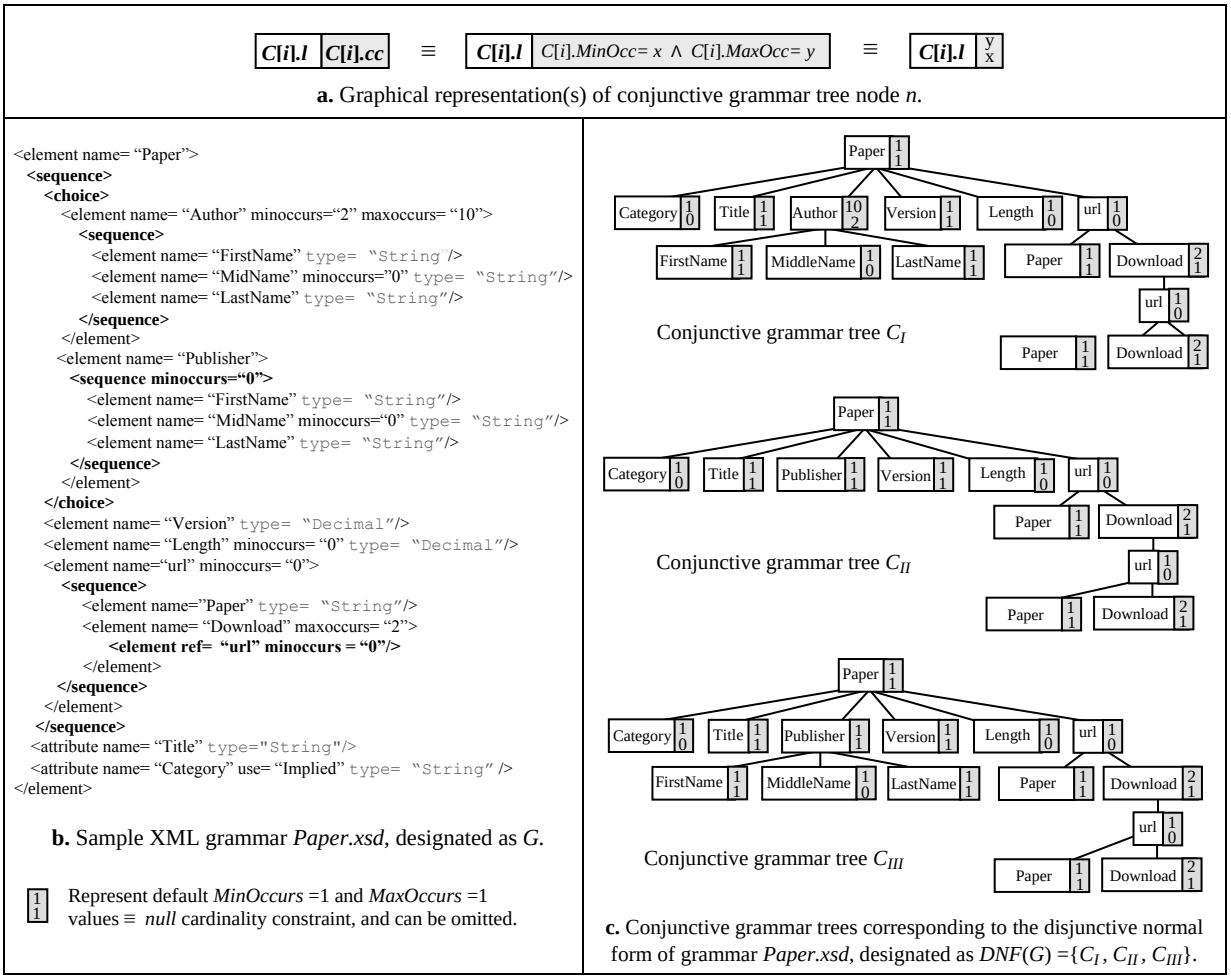


Fig. 7. Sample conjunctive grammar tree representations.

The *DNF* form of *Paper.xsd*, unfolded in a set of conjunctive trees, is shown in Fig. 7.c. Grammar *Paper.xsd* is first run through our transformation rules (cf. Fig. 8). Then, the resulting flattened grammar, encompassing *Or* operators, is represented as three conjunctive grammar trees (Fig. 7.b and c) underlining the three structural configurations that can be obtained following the different combinations of the *Or* operators.

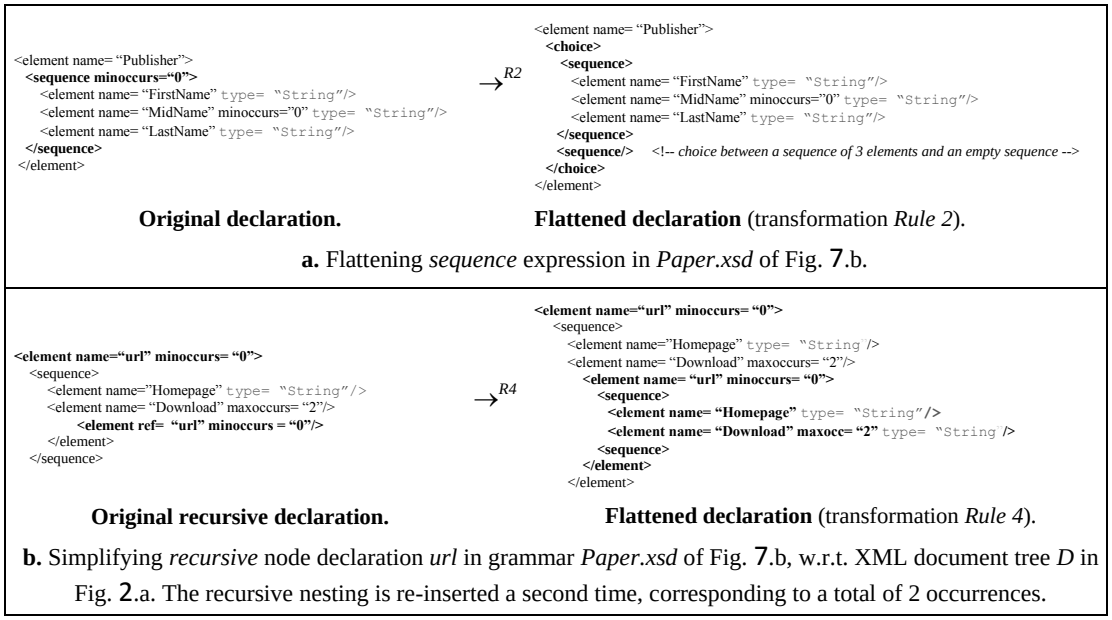


Fig. 8. Flattening sequence and recursive declarations in grammar *Paper.xsd* of Fig. 7.b.

Note that in addition to handling more expressive XSD constraints (*MinOccurs* and *MaxOccurs*), as well as alternative expressions and recursive declarations, our XML grammar tree model also straightforwardly handles *context-sensitive* XSD declarations, where identically labeled elements can have multiple definitions in different contexts in the grammar (as opposed to *context-free* DTD declarations). For instance, grammar *Paper.xsd* contains two elements shaving the same label *Paper*: i) the grammar root node element, and ii) the first child of element *url*.

Hence with XML documents and grammars represented as rooted ordered labeled trees, the problem of XML document/grammar structural comparison now comes down to comparing corresponding trees.

4. XML Document and Grammar Tree Comparison

As mentioned previously, our approach consists of two main phases: i) *Tree Representation* of documents and grammars as rooted ordered labeled trees (described in the previous section), ii) and *Tree Edit Distance Comparison* for computing the similarity between document and grammar tree structures. The overall algorithm is presented in Fig. 9.

After transforming the XML document and XML grammar into their tree representations (Fig. 9, lines 1-2), the edit distance between the document tree and each conjunctive grammar tree is computed (lines 3-10).

Definition 17 – Tree Edit Distance (*TED*): The *edit distance* between two trees *A* and *B* is defined as the minimum cost of all edit scripts that transforms *A* to *B*, $TED(A, B, Cost_{Op})=Min\{Cost_{ES}\}$, noted as $TED(A, B)$ [12, 82] ●

Hence, the problem of comparing two trees *A* and *B*, i.e., evaluating the structural similarity between *A* and *B*, is viewed as the problem of computing corresponding tree edit distance, i.e., minimum cost edit script [82]. In this context, the notion of edit distance can be adapted to our study as follows:

Definition 18 – TED_{XDoc_XGram}: Given an XML document tree *D*, a conjunctive grammar tree *C*, as well as corresponding tree edit operations costs, denoted $Cost_{InsTree/DelTree}$, and based on the traditional definition of tree edit distance, we define $TED_{XDoc\_XGram}(D, C, Cost_{InsTree/DelTree})$, noted simply as $TED_{XDoc\_XGram}(D, C)$, as the minimum cost of all edit scripts transforming *D* into a document tree *D'* which is valid w.r.t. *C*, i.e., $C \vDash D'$ ●

```
Algorithm XDoc_XGram_Comparison
Input:  D          // XML document
        G          // XML grammar
        {R}        // Set of transformation rules (cf. Tables 1 and 2)
Output: Sim(D, G)  // Structural similarity value between D and G ∈ [0,1]

Begin
  Begin Tree Representation phase
    D_Tree = XDoc_to_Tree(D)                    // Document tree representation      1
    G_TreeSet = XGram_to_Tree(G, D, {R})        // Grammar tree representation       2
  End Tree Representation phase
  Begin Tree Edit Distance phase
    Dist[] = new [1… |G_TreeSet|]               // |G_TreeSet| ≡ |{C}_G|, n# of conjunctive   3
                                                // grammar trees representing G      4
    Multi-thread (i=1, i ≤ |G_TreeSet|, i++)    // Tree edit distance multi-threading   5
    {                                                                                 6
      {Cost_InsTree/DelTree} = TOC_XDoc(D_Tree) ∪ TOC_XGram(G_TreeSet[i])  // Tree operations costs  7
      Dist[i] = TED_XDoc_XGram(D_Tree, G_TreeSet[i], {Cost_InsTree/DelTree})  // Tree edit distance  8
    }                                                                                 9
    Return Sim_XDoc_XGram(D, G) = Max_{1≤ i ≤ |G_TreeSet|} { 1 / (1 + Dist[i]) }   // Structural Similarity  10
  End Tree Edit Distance phase
End
```

Fig. 9. Pseudo-code of overall XML document/grammar comparison algorithm.

The comparison is undertaken using concurrent computing, i.e., multi-threading, evaluating the similarity between the document tree and each of the conjunctive grammar trees simultaneously (lines 5-9) since they constitute a forest of separate tree structures (corresponding to the input grammar) following our grammar tree model, without neither interfering nor relying on each other's results. Tree edit operations costs are computed (via algorithms TOC_{XDoc} and TOC_{XGram}, line 7, mentioned in the following section), and are consequently provided as input to the main tree edit distance algorithm (TED_{XDoc_XGram}, line 8)[1]. An *atomic* edit operation on a tree is either the insertion (addition) of an inner node, the insertion of a leaf node, the deletion (removal) of an inner node, the deletion of a leaf node, or the replacement (i.e., update) of a node by another one. A *complex* tree edit operation is a sequence of atomic tree edit operations, treated as one single operation, such as the insertion/deletion of a whole sub-tree, or the relocation (moving)

[1] Algorithms TOC_{XGram} and TED_{XDoc_XGram} are developed in the following sections. Algorithm TOC_{XDoc} is provided in [73].

of a sub-tree from one position into another in its containing tree[1]. A sequence of edit operations, called an *edit script*, $ES=\prec op_1, \ldots, op_k \succ$ can be applied on a tree T, producing a resulting tree T' by applying the edit operations $op_1, \ldots op_k$ in ES to T, following their order of appearance in the script. By associating costs, $Cost_{Op}$, with edit operations, the cost of an *edit script* is defined as the sum of the costs of its component operations [12, 18]: $Cost_{ES} = \sum_{i=1}^{|ES|} Cost_{Op_i}$.

Then, the maximum similarity (minimum distance) between the document tree and each conjunctive grammar tree is evaluated as the overall document/grammar structural similarity value (line 10). One can realize, based on the definition of TED_{XDoc_XGram}, that our approach allows both:

- *Exact* document/grammar structure validation (Definition 5), where $TED_{XDoc\_XGram}(D, C) = 0 \Rightarrow C \vDash D$,
- *Approximate* document/grammar validation (cf. Definition 6), where $TED_{XDoc\_XGram}(D, C) \neq 0 \Rightarrow C \mid\approx_{\sigma} D$, such that σ designates a similarity value which is inversely proportional to $TED_{XDoc\_XGram}(D, C)$: the lesser the similarity σ, the larger the distance $TED_{XDoc\_XGram}(D, C)$, i.e., the larger the edit script cost needed to transform D into a document tree D' such that $C \vDash D'$.

As for the method to compute $TED_{XDoc\_XGram}(D, C)$, we build on a dynamic programming formulation similar to a central tree edit distance algorithm by Nierman and Jagadish in [48], mainly in terms of the edit operations utilized. However, we introduce novel recurrences specifically designed to handle XML grammar cardinality constraints (namely *MinOccurs* and *MaxOccurs*).

In the reminder, we first discuss tree edit operations costs in Section 4.1, and subsequently develop the main algorithm and similarity measure in Sections 4.2 and 4.3. Section 4.4 presents computation examples. Time and space complexity analyses are provided in Section 4.5.

4.1. Tree Edit Operations Costs: TOC_{XDoc} & TOC_{XGram}

Our tree edit distance algorithm employs five edit operations: i) *leaf node insertion*, ii) *leaf node deletion*, iii) *node update*, iv) *tree insertion* and v) *tree deletion* adopted from [18, 48] (formal definitions are provided in Appendix III). However, a central issue in most edit distance approaches is how to determine edit operations cost values, in order to consequently determine the edit distance value (i.e., the minimum cost of all possible edit scripts). An intuitive way would be to assign identical unit costs to single node operations:

$$\begin{gathered} \text{Cost}_{\text{Ins}}(x) = \text{Cost}_{\text{Del}}(x) = 1 \\ \text{Cost}_{\text{Upd}}(x, y) = \text{Cost}_{\text{Upd}}(x.\ell, y.\ell) = 1 \text{ when } x.\ell \neq y.\ell, \text{ otherwise, } \text{Cost}_{\text{Upd}} = 0, \\ \text{underlining that no changes are to be made to the label of node } x. \end{gathered} \quad \textbf{(1)}$$

As for tree deletion (insertion) operations, they can be naturally evaluated as the sum of the costs of deleting (inserting) all individual nodes in the considered sub-tree [26], such as:

$$\text{Cost}_{\text{DelTree/InsTree}}(T) = \sum_{\text{All nodes } x_i \in T} \text{Cost}_{\text{Del/Ins}}(x_i) \quad \textbf{(2)}$$

Following our approach, computing $TED_{XDoc\_XGram}(D, C)$ comes down to transforming document tree D into D' to obtain $C \vDash D'$. To do so, node/tree deletion operations will be applied on the document tree D to remove those nodes which do not conform to the grammar, whereas node/tree insertion operations will add grammar nodes to the document tree D in order to obtain $C \vDash D'$. Yet, recall that nodes in grammar trees are associated cardinality constraints: *MinOccurs* and *MaxOccurs*, specifying the allowed number of occurrences corresponding to a (sub-tree rooted at a) given node. Hence, grammar tree insertion operations costs are updated accordingly in order to evaluate TED_{XDoc_XGram}:

- *Case 1 – Optional Grammar Nodes*: An optional grammar tree node $x_i \in C$ such as $x_i.MinOccurs = 0$, ($\forall\ x_i.MaxOcc$), along with its sub-tree C_i (i.e., the sub-tree rooted at $x_i = R(C_i)$), do not affect the costs of tree insertion operations applied on C. In other words, node/sub-tree x_i/C_i do not have to be inserted in the document tree D to obtain $C \vDash D'$, and hence should not affect edit distance cost.
- *Case 2 – Mandatory Grammar Nodes*: A mandatory and/or repeatable grammar tree node $x_i \in C$ such as $x_i.MinOccurs \neq 0$ ($\forall\ x_i.MaxOccurs$), along with its sub-tree C_i (where $R(C_i) = x_i$), affect the costs of tree insertion operations applied on C, considering the minimum number of occurrences required for x_i (C_i), i.e., $min\{x_i.MinOccurs, x_i.MaxOccurs\} \equiv x_i.MinOccurs$. In other words, when x_i/C_i is mandatory/repeatable, then it should occur (or should be inserted) in the document tree D, a minimum number of times ($x_i.MinOccurs$) necessary to obtain $C \vDash D'$, thus affecting tree insertion operations costs accordingly.

Formally, given a conjunctive XML grammar tree C (with root node $R(C)$ of degree k), and its first level sub-trees $C_1, \ldots, C_k$ (i.e., the sub-trees rooted at the children nodes of $R(C)$), we compute corresponding tree operations costs as:

---

[1] Other complex operations such as sub-tree copying and gluing have been considered [19]. 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.

$$\text{Cost}_{\text{InsTree}}(C) = \text{Cost}_{\text{Ins}}(R(C)) + \sum_{\text{All first-level sub-trees } C_i \text{ of } C} \text{Cost}_{\text{InsTree}}(C_i) \times R(C_i).MinOcc \quad \textbf{(3)}$$

where $R(C_i).MinOccurs$ underlines the *MinOccurs* constraints associated to the root of C_i.

The algorithm for computing insert tree operations costs is provided in Fig. 10. Here, we only develop XML grammar tree processing, TOC_{XGram}, and omit the pseudo-code for XML document tree processing, TOC_{XDoc} (computing tree deletion operations costs) since the latter is straightforward following formula (2). Algorithm TOC_{XGram} goes through all sub-trees of the conjunctive XML grammar tree, computing grammar sub-tree insertion operations costs following Formula (3) (Fig. 10, lines 9-12), taking into account corresponding sub-tree node *MinOccurs*.

```
Algorithm TOC_XGram
Input:  C                          // XML conjunctive grammar tree
Output: {Cost_InsTree}_C           // Tree insertion operations costs, for all sub-trees in C

Begin
    M = Degree(C)                  // The number of first level sub-trees in conjunctive grammar tree G.   1
    Cost_InsTree (C) = Cost_Ins(R(C))   // Initializing grammar sub-tree costs                             2
                                   // with the cost of corresponding sub-tree root node.
    If (M = 0)                                                                                            3
    {                                                                                                     4
      Return Cost_Ins(R(C))        // Leaf node operations are assigned unit costs                       5
    }                              // in our approach (basic cost model),                                6
    Else                                                                                                  7
    {                                                                                                     8
      For (i = 1 ; i ≤ M ; i++)    // Going through the first level sub-trees of C, C_i / i=1…M          9
      {                                                                                                   10
        Cost_InsTree(C) = Cost_InsTree(C) + (TOC_XGram(C_i) × R(C_i).MinOccurs)                           11
      }                                                                                                   12
    }                                                                                                     13
    Return {Cost_InsTree}_C        // Tree insertion operations costs                                    14
End
```

Fig. 10. Algorithm TOC_{XGram} for computing XML grammar sub-tree operations costs.

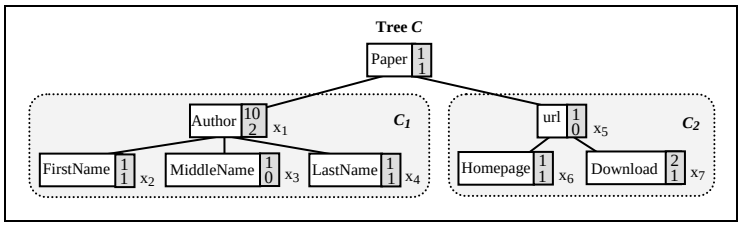


Fig. 11. Sample conjunctive grammar tree *C* (extracted from first grammar tree in Fig. 7.c).

Consider sample grammar tree *C* in Fig. 11. Tree insertion operations costs following TOC_{XGram} are computed as:

- $\text{Cost}_{\text{InsTree}}(C_1) = \text{Cost}_{\text{Ins}}(R(C_1)) + [\ \text{Cost}_{\text{Ins}}(x_2) + \text{Cost}_{\text{Ins}}(x_3) \times 0 + \text{Cost}_{\text{Ins}}(x_4)\ ] = 3$
- $\text{Cost}_{\text{InsTree}}(C_2) = \text{Cost}_{\text{Ins}}(R(C_2)) + [\ \text{Cost}_{\text{Ins}}(x_6) + \text{Cost}_{\text{Ins}}(x_7)\ ] = 3$
- $\text{Cost}_{\text{InsTree}}(C) = \text{Cost}_{\text{Ins}}(R(C)) + [\ \text{Cost}_{\text{InsTree}}(C_1) \times 2 + \text{Cost}_{\text{InsTree}}(C_2) \times 0\ ] = 7$

Here, the cost of inserting sub-tree C_1 rooted at the node of label *Author* (x_1) is equal to 3. This is because node $x_3.MinOcc$=0, which means that the occurrence of node *MiddleName* is not required (in the transformed document tree for it to conform to the grammar). In turn, the cost of inserting tree *C* (as a whole) is equal to 7, since its first-level sub-tree C_1 is required to appear a minimum number of 2 times in the transformed document tree ($R(C_1).MinOccurs = x_1.MinOccurs = 2$, yielding $Cost_{DelTree}(C_1) \times 2$), whereas sub-tree C_2 is optional (yielding $Cost_{DelTree}(C_2) \times 0$).

Note that both *MinOccurs* and *MaxOccurs* constraints are used in our main tree edit distance algorithm, to verify whether the minimum/maximum allowed number of occurrences for a given grammar node (sub-tree) are violated in the document tree being compared, so as to allocate tree edit operations accordingly (described in the following section).

In this study, we restrict our presentation to the basic cost schemes above, since we focus on the structural properties of XML documents and grammars (i.e., considering parent/child relationships and ordering among XML elements, identified by their labels). The investigation of alternative tree operations cost models (considering for instance the semantic relatedness between document and grammar node labels given a semantic reference such as WordNet [45], Wikipedia [84], or Google [37]) will be addressed in a dedicated upcoming study.

4.2. Tree Edit Distance (TED) Algorithm: TED_{XDoc_XGram}

As briefly mentioned previously, we propose a novel tree edit distance method to consider the structural properties of XML document trees and conjunctive grammar trees (inspired by existing tree edit distance proposals, namely [26, 48]). Hereunder, we first describe the overall process of our main algorithm. Then, we present the *Traditional TED*

Recurrence formulation as the basic foundation of our algorithm, and introduce our *Extended TED Recurrence* formulation taking into account the *Minoccurs* and *MaxOccurs* constraints. We then develop computation examples.

4.2.1. Main Algorithm

The overall algorithm TED_{XDoc_XGram} for computing the edit distance between an XML document tree D and a conjunctive grammar tree C is shown in Fig. 12. It builds on an *Extended TED Recurrence* to identify the minimum cost edit script (i.e., the minimum distance, thus maximum similarity) transforming D into D' to obtain $C \vDash D'$.

In short, algorithm TED_{XDoc_XGram} recursively goes through the sub-trees of both XML document and XML grammar tree structures, combining node update, tree insertion and tree deletion operations so as to identify the sequence of operations (edit script) of minimal cost. The insertion/deletion of single nodes is undertaken via *tree insertion* and *tree deletion* operations applied on leaf node sub-trees. In other words, *leaf node insertion/deletion* operations do not contribute directly to the edit distance algorithm, but are utilized in computing tree insertion and tree deletion operations costs (cf. TOC_{XGram} in Fig. 10).

First, the *update* operation is applied to the roots of the sub-trees being compared (relabeling sub-tree root nodes, Fig. 12, line 6). Then, *tree deletion* operations are applied to corresponding document first-level sub-trees (line 7), and tree *insertion operations* are applied on grammar first-level sub-trees taking into account the *MinOccurs* constraint (line 8), in order to consider the minimum number of occurrences required in the document tree so as to conform to the grammar tree (as discussed with sub-tree operations costs in Section 4.1). Consequently, the edit distance process TED_{XDoc_XGram} is recursively called once for each pair of sub-trees D_i and C_j occurring at the same structural level (depth) in the document and grammar trees being compared. This is undertaken following our *Extended TED Recurrence* formula (lines 11-24) described in detail the following section. The minimum distance between all sub-trees (first-level, second-level, and so on) of the document tree D and grammar tree C is finally returned (line 28). When the grammar tree is free of constraint operators (i.e., when all elements in C are associated default constraints *MinOccurs* = *MaxOccurs* = 1), our algorithm simplifies to a classical TED process (namely the algorithm in [48]).

Algorithm TED_{XDoc_XGram}

Input: D // XML document tree
C // Conjunctive grammar tree
$\{Cost_{InsTree/DelTree}\}$ // Tree operations costs computed via TOC_{XDoc} and TOC_{XGram}

Output: $TED_{XDoc\_XGram}(D, C)$ // Edit distance between D and C

Begin

M = Degree(D) // The number of first level sub-trees in D 1
N = Degree(C) // The number of first level sub-trees in C 2
Dist [,] = new [0...M, 0…N] 3
NbOcc [] = new [0…N] // Keeping track of the number of occurrences for each grammar sub-tree, 4
NbOcc [0...N] = 0 // in order to handle corresponding *MinOccurs* and *MaxOccurs* constrains 5
Dist [0, 0] = $Cost_{Upd}(R(D).\ell, R(C).\ell)$ // $R(D).\ell$ and $R(C).\ell$ are the labels of the roots of trees D and C 6
For (i = 1 ; i ≤ M ; i++) { Dist[i, 0] = Dist[i-1, 0] + $Cost_{DelTree}(D_i)$ } 7
For (j = 1 ; j ≤ N ; j++) { Dist [0, j] = Dist [0, j-1] + $Cost_{InsTree}(C_j) \times R(C_j).MinOccurs$ } 8
For (j = 1 ; j ≤ N ; j++) 9
{ 10
For (i = 1 ; i ≤ M ; i++) 11
{
NbOcc[j]++ 12
α = Dist[i-1, j] + $Cost_{DelTree}(D_i)$ // Considering sub-tree deletion costs 13
β = Dist[i, j-1] + $Cost_{InsTree}[C_j] \times R(C_j).MinOccurs$ // Considering sub-tree insertion costs 14
If (NbOcc[j] < $R(C_j)$.MaxOccurs) 15
{ 16
If (NbOcc[j] < $R(C_j)$.MinOccurs) // Considering the *MinOccurs* constraint, γ_1 17
{ $\gamma = Dist[i - NbOcc[j], j-1] + \sum_{n=0}^{NbOcc[j]-1} TED_{XDoc\_XGram}(D_{i-n}, C_j, \{Cost_{InsTree/DelTree}\}) + Cost_{InsTree}(C_j) \times (R(C_j).MinOccurs - NbOcc[j])$ } 18
Else // Considering the *MaxOccurs* constraint, γ_2 19
{ $\gamma = Dist[i - NbOcc[j], j-1] + \sum_{n=0}^{NbOcc[j]-1} TED_{XDoc\_XGram}(D_{i-n}, C_j, \{Cost_{InsTree/DelTree}\})$ }
20
}
Else 21
{ γ = Dist[i-1, j-1] + $TED_{XDoc\_XGram}(D_i, C_j, \{Cost_{InsTree/DelTree}\})$ } // Classic edit distance formula, γ_3 22
Dist[i, j] = min{ α, β, γ } // Identifying minimum distance 23
If(Dist[i, j] = α || Dist[i, j] = β) { NbOcc[j] = 0 } // Updating *NbOcc* value corresponding to sub-tree C_j 24
} // End For i 25
} // End For j 26
Return Dist[M, N] // Edit distance value 27
End 28

Fig. 12. Algorithm TED_{XDoc_XGram} for comparing an XML document tree and a conjunctive grammar tree.

4.2.2. TED Recurrences

$TED_{XDoc\_XGram}(D, C)$ computes, as sub-routines, the edit distance between pairs of first-level sub-trees $D_i \in D$ and $C_i \in C$ (i.e., the sub-trees rooted at the children nodes of $R(D)$ and $R(C)$ respectively), noted $TED_{XDoc\_XGram}(D_i, C_j)$. We use left-to-right numbering to identify first-level sub-tree order. We denote by $Dist[i, j]$ the edit distance matrix keeping track of the edit distance between document tree D with only its i first-level sub-trees, identified as *partial document tree* $D<i>$, and grammar tree C with only its j first-level sub-trees, identified as *partial grammar tree* $C<j>$.

Hence, a traditional tree edit recurrence adopted from existing approaches [26, 48] can be represented as:

Traditional TED Recurrence. Consider the pair of first-level sub-trees $D_i \in D$ and $C_i \in C$. Then:

$$\text{Dist}[i, j] = \min \begin{cases} \alpha = \text{Dist}[i-1, j] + \text{Cost}_{\text{DelTree}}(D_i) \\ \beta = \text{Dist}[i, j\text{-}1] + \text{Cost}_{\text{InsTree}}(C_j) \\ \gamma = \text{Dist}[i\text{-}1, j\text{-}1] + TED_{XDoc\_XGram}(D_i, C_j) \end{cases}$$

Proof. Our goal is to find the minimum cost script transforming $D<i>$ into a partial document tree $D<i>'$ such that $C<j> \vDash D<i>'$. This can be computed in three ways:

- Having $Dist[i\text{-}1, j]$, we spend $\alpha = Dist[i\text{-}1, j] + \text{Cost}_{\text{DelTree}}(D_i)$, deleting D_i from $D<i>$
- Having $Dist[i, j\text{-}1]$, we spend $\beta = \text{Dist}[i, j\text{-}1] + \text{Cost}_{\text{InsTree}}(C_j)$, inserting one occurrence of C_j into $D<i>$
- Having $Dist[i\text{-}1, j\text{-}1]$, we spend $\gamma = Dist[i\text{-}1, j\text{-}1] + TED_{XDoc\_XGram}(D_i, C_j)$, transforming D_i into D_i' such that $C_j' \vDash D_i'$.

Since these three cases express all possible tree edit paths yielding $\text{Dist}[i, j]$, we keep the *minimum* from these costs.

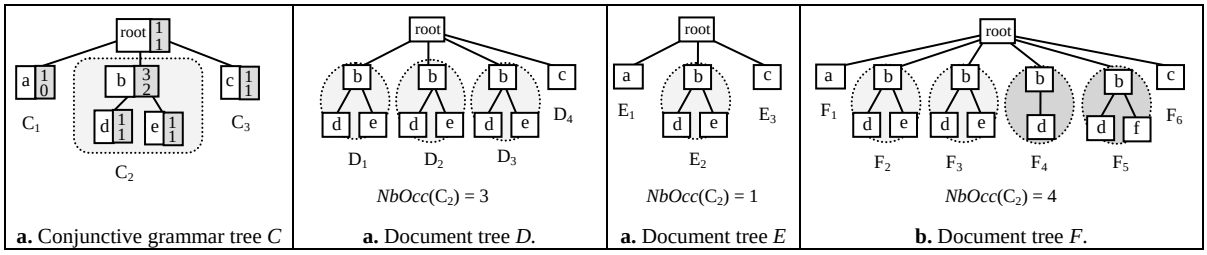
Proof description. *Traditional TED Recurrence* carries from Nierman & Jagadish's approach [48]. The minimum cost script transforming partial tree $D<i>$ into $D<i>'$ in order to have $C<j> \vDash D<i>'$ can be computed in three ways:

- Having $Dist[i\text{-}1, j]$ and the cost of deleting sub-tree D_i, we need to spend at least $Dist[i\text{-}1, j] + \text{Cost}_{\text{DelTree}}(D_i)$ to transform partial document tree $D<i>$ into $D<i>'$ in order to obtain $C<j> \vDash D<i>'$.
- Having $Dist[i, j\text{-}1]$ and the cost of inserting sub-tree C_j, we need to spend at least $\text{Dist}[i, j\text{-}1] + \text{Cost}_{\text{InsTree}}(C_j)$ to transform $D<i>$ into $D<i>'$ to obtain $C<j> \vDash D<i>'$.
- Having $Dist[i\text{-}1, j\text{-}1]$, we need to spend at least $Dist[i\text{-}1, j\text{-}1] + TED_{XDoc\_XGram}(D_i, C_j)$ to transform $D<i>$ into $D<i>'$ to obtain $C<j> \vDash D<i>'$.

Since the three cases above express all possible tree edit paths following the set of edit operations considered in our approach, hence we keep in $\text{Dist}[i, j]$ the minimum from the three costs α, β, and γ, i.e., the minimum attainable distance between $D<i>$ and $C<j>$ allowing to transform $D<i>$ into $D<i>'$ such that $C<j> \vDash D<i>'$ □

TED_{XDoc_XGram} is recursively applied on all sub-trees in D and C (first-level, second-level, so on, cf. γ computation, following [48]), hence identifying the minimum cost scrip transforming D into a document tree D' such that $C \vDash D'$. In short, the *Traditional TED Recurrence* underlines the most basic case where the conjunctive grammar tree C is virtually free of cardinality constraint operators (i.e., when all elements in C are associated default constraints *MinOccurs* = *MaxOccurs* = 1), such that all elements (sub-trees) are mandatory and should appear exactly once.

Hence, we propose an *Extended TED Recurrence* to specifically consider the *MinOccurs* and *MaxOccurs* cardinality constraints when comparing document and grammar trees. To do so, we keep track of the number of occurrences *NbOcc* of the document sub-trees corresponding to each grammar sub-tree in the conjunctive grammar at hand (which will allow us to verify whether the corresponding grammar sub-tree *MinOccurs*/*MaxOccurs* constraint has been met, or violated, and if so to what extent). Sub-tree occurrences in the document tree can be exact (conforming, $\vDash$) or approximate (similar enough, $|\approx$) to the grammar sub-tree.



a. Conjunctive grammar tree C | **a.** Document tree D. | **a.** Document tree E | **b.** Document tree F.

Document tree D is valid w.r.t. C ($C \vDash D$), whereas document trees E and F are similar (but not valid) w.r.t. C ($C \; |\approx \{E, F\}$).

Fig. 13. Sample document and grammar trees.

Consider the example in Fig. 13, comprising of conjunctive grammar tree C and document trees D, E, and F. Here, one can realize that grammar sub-tree C_2 occurs three times in document tree D, where $C_2 \vDash \{D_1, D_2, D_3\}$. Also, grammar sub-tree C_2 occurs once in document tree E, where $C_2 \vDash E_2$. Yet, sub-tree C_2 occurs 4 times in document tree

F, where $C_2 \models \{F_2, F_3\}$ whereas $C_2 \mid\approx \{F_4, F_5\}$. Note that we can computationally decide whether a document sub-tree D_i consists of an exact ($\models$) or approximate ($\mid\approx$) occurrence of a grammar sub-tree C_j based on the corresponding edit distance score $TED_{XDoc\_XGram}(D_i, C_j)$ (e.g., $TED_{XDoc\_XGram}(D_i, C_j)=0$ is obtained when $C_j \models D_i$, whereas $TED_{XDoc\_XGram}(D_i, C_j) \neq 0$ is obtained when $C_j \mid\approx D_i$).

Hence, based on the *Traditional TED Recurrence*, and the notion of number of occurrences: *NbOcc***,** we can effectively consider the *MinOccurs* and *MaxOccurs* constraints in our tree edit distance computations as follows:

Extended TED Recurrence (TED$^+$). Consider the pair of first-level sub-trees $D_i \in D$ and $C_i \in C$. Let *NbOcc*[j] be a special counter keeping track of the number of occurrences (exact/approximate matches) of grammar sub-tree C_j in the document tree D. Then, considering $R(C_i).MinOccurs$ and $R(C_i).MaxOccurs$:

$$\text{Dist}[i, j] = \min \begin{cases} a = \text{Dist}[i-1, j] + \text{Cost}_{\text{DelTree}}(D_i) \\ b = \text{Dist}[i, j-1] + \text{Cost}_{\text{InsTree}}(C_j) \times R(C_j).MinOccurs \\ g = \begin{cases} g_1 & \text{if } (\text{NbOcc[j]} < R(C_j).MinOccurs) \quad // \textbf{ Condition 1 :} \text{ Considering MinOccurs constraint} \\ g_2 & \text{else if } (\text{NbOcc[j]} < R(C_j).MaxOccurs) \quad // \textbf{ Condition 2 :} \text{ Considering MaxOccurs constraint} \\ g_3 & \text{else} \quad // \text{ Traditional TED computation} \end{cases} \end{cases}$$

Where:

$$\gamma_1 = \text{Dist}[i - \text{NbOcc[j]}, j-1] + \sum_{n=0}^{\text{NbOcc[j]}-1} TED_{XDoc\_XGram}(D_{i-n}, C_j) + \text{Cost}_{\text{InsTree}}(C_j) \times (R(C_j).MinOccurs - \text{NbOcc[j]})$$

$$g_2 = \text{Dist}[i - \text{NbOcc[j]}, j-1] + \sum_{n=0}^{\text{NbOcc[j]}-1} TED_{XDoc\_XGram}(D_{i-n}, C_j)$$

$$g_3 = \text{Dist}[i-1, j-1] + TED_{XDoc\_XGram}(D_i, C_j)$$

Proof. Our goal is to find the minimum cost edit script transforming $D\langle i\rangle$ into $D\langle i\rangle'$ to obtain $C\langle j\rangle \models D\langle i\rangle'$, given that a grammar sub-tree $C_i \in C\langle j\rangle$ is required to appear a minimum number of times ($R(C_j).MinOccurs$) and a maximum number of times ($R(C_j).MaxOccurs$) in $D\langle i\rangle'$. This can be computed in three ways:

- Having $Dist[i\text{-}1, j]$, we spend $\alpha = Dist[i\text{-}1, j] + \text{Cost}_{\text{DelTree}}(D_i)$, deleting D_i from $D\langle i\rangle$
- Having $Dist[i, j\text{-}1]$, we spend $\beta = Dist[i, j\text{-}1] + \text{Cost}_{\text{InsTree}}(C_j) \times R(C_j).MinOccurs$, inserting sub-tree C_j into $D\langle i\rangle$ as many times as required by the corresponding $R(C_j).MinOccurs$ constraint.
- Having $Dist[i\text{-}1, j\text{-}1]$, we need to account for three alternative cost factors:
  - Having $NbOcc[j]$ occurrences of C_j in $D\langle i\rangle$, such that $NbOcc[j] \in [R(C_j).MinOccurs, R(C_j).MaxOccurs]$ (Condition 2) we compute the edit distance between all sub-trees in $D\langle i\rangle$ which match C_j, starting from $D_{i - NbOcc[j]}$ (first match) to D_i (last match):
    $\gamma_2 = \text{Dist}[i\text{-}NbOcc[j], j\text{-}1] + \sum_{n=0}^{\text{NbOcc[j]}-1} TED_{XDoc\_XGram}(D_{i-n}, C_j)$
  - Having $NbOcc[j] < R(C_j).MinOccurs$ (Condition 1) we add to the costs of the $NbOcc[j]$ existing occurrences of C_j (i.e., γ_2) the cost of additional sub-tree occurrences required to occur in $D\langle i\rangle$ in order to fulfill C_j's *MinOccurs* constraint: $\gamma_1 = \gamma_2 + \text{Cost}_{\text{InsTree}}(C_j) \times (\text{R}(C_j).MinOccurs - NbOcc[j])$.
  - Otherwise, when $NbOcc[j] > R(C_j).MaxOccurs$, then we apply the *Traditional TED Recurrence* factor γ_3, such that every additional C_j occurrence is treated as any regular sub-tree occurrence.

Since these three cases express all possible tree edit paths yielding $\text{Dist}[i, j]$, we keep the *minimum* from these costs.

Proof description. Our goal is to find the minimum cost edit script transforming partial document tree $D\langle i\rangle$ into $D\langle i\rangle'$ to obtain $C\langle j\rangle \models D\langle i\rangle'$, considering that a grammar (element) sub-tree $C_i \in C\langle j\rangle$ is required to appear a minimum number of times ($R(C_j).MinOccurs$) and a maximum number of times ($R(C_j).MaxOccurs$) in the transformed partial document tree $D\langle i\rangle'$ for it to conform to $C\langle j\rangle$. This can be computed in three ways:

- Having the value $Dist[i\text{-}1, j]$ and the cost of (deleting) sub-tree D_i, we need to spend at least $Dist[i\text{-}1, j] + \text{Cost}_{\text{DelTree}}(D_i)$ to transform the partial document tree $D\langle i\rangle$ into $D\langle i\rangle'$ to obtain $C\langle j\rangle \models D\langle i\rangle'$. This carries from the *Traditional TED Recurrence*.
- Having the value $Dist[i, j\text{-}1]$ and the cost of (inserting) sub-tree C_j, we need to spend at least $Dist[i, j\text{-}1] + \text{Cost}_{\text{InsTree}}(C_j) \times R(C_j).MinOccurs$ to transform $D\langle i\rangle$ into $D\langle i\rangle'$ to obtain $C\langle j\rangle \models D\langle i\rangle'$. This requires inserting sub-tree C_j into $D\langle i\rangle$, as many times as required by the corresponding $R(C_j).MinOccurs$ constraint.

– Having the value *Dist*[*i*-1, *j*-1], we need to account for two cost factors: the costs of i) *existing sub-tree occurrences* of C_j (sub-trees matching C_j which already appear in the partial document tree), and ii) *remaining sub-tree occurrences* of C_j (sub-trees which are still required to appear – which need to be inserted – in the partial document tree) to fulfill C_j's *MinOccurs* and *MaxOccurs*' constraints:

- ***Existing sub-tree occurrences*** cost: It is only applied when $NbOcc[j] < R(C_j).MinOccurs$, i.e., when the number of sub-trees matching C_j in the partial document tree, does not yet fulfill $R(C_j).MinOccurs$ (cf. γ_2). Having $NbOcc[j]$ existing occurrences (exact/approximate match candidates) of sub-tree C_j in partial document tree $D\langle i\rangle$, the similarity score between each of the (exact/approximate) match candidates on one hand, and C_j on the other hand, need to be computed, in order to identify the overall cost of these *existing sub-tree occurrences*. To do so, we need to start from Dist[*i-NbOcc*[*j*], *j*-1], the distance value at the last position (in the edit distance table) preceding the occurrence of the first document sub-tree match candidate with C_j, i.e., $D_{i\,-\,NbOcc[j]}$, and then spend at least $\sum_{n=0}^{NbOcc[j]-1} TED_{XDoc\_XGram}(D_{i-n}, C_j)$, covering the sum of the tree edit distance costs for comparing grammar sub-tree C_j with all consecutive first-level document sub-trees in $D\langle i\rangle$ ranging from sub-tree $D_{i\,-\,NbOcc[j]}$ (the first exact/approximate match candidate of grammar sub-tree C_j in the document tree) to D_i (the last exact/approximate match candidate of C_j in the document tree).

  Example: Consider computing the edit distance between document tree F and grammar tree C from Fig. 13. Evaluating the γ factor at *Dist*[4, 2], considering grammar sub-tree C_2, and having *NbOcc*[2]=3 (given that 3 possible candidate sub-trees matching C_2 have been identified: F_2, F_3, and F_4) yields: $Dist[0, 1] + TED_{XDoc\_XGram}(F_2, C_2) + TED_{XDoc\_XGram}(F_3, C_2) + TED_{XDoc\_XGram}(F_4, C_2) = 0 + 0 + 0 + 1$. This means that $C_2 \vDash \{F_2, F_3\}$ such that their occurrences in partial tree $D\langle 3\rangle$ do not entail any additional cost, whereas $C_2 \mid\!\approx F_4$ requiring a transformation of cost=1 (i.e., the insertion of node e in F_4) for partial document tree $F\langle 4\rangle$ to become valid w.r.t. grammar tree $C\langle 2\rangle$, i.e., $C\langle 2\rangle \vDash F\langle 4\rangle'$ (cf. graphical presentation in Fig. 14.c, and more detailed computation examples in the following section).

- ***Remaining sub-tree occurrences*** cost: It is only applied when $NbOcc[j] \in [R(C_j).MinOccurs, R(C_j).MaxOccurs)]$, i.e., when the number of sub-trees matching C_j in the partial document tree, remains within C_j's constraints margin (cf. γ_1). Here, in addition to the edit distance costs of the *NbOcc*[*j*] existing occurrences (exact/approximate matches) of sub-tree C_j in the partial document tree $D\langle i\rangle$, we need to account for the cost of sub-tree occurrences (corresponding to C_j) which have not yet been inserted in $D\langle i\rangle$ but which are required to occur in $D\langle i\rangle$, to fulfill the *MinOccurs* constraint associated to C_j, in order to obtain $C\langle j\rangle \vDash D\langle i\rangle'$. This is mathematically concretized in the edit distance formula by adding: $Cost_{InsTree}(C_j)\times(R(C_j).MinOccurs - NbOcc[j])$, covering the cost of inserting sub-tree C_j multiplied by the minimum number of occurrences needed $R(C_j).MinOccurs$, minus the number of already existing occurrences $NbOcc[j]$ of exact/approximate matches of C_j in the partial document tree $D\langle i\rangle$. The *remaining sub-tree occurrences* factor is (naturally) disregarded when $R(C_j).MinOccurs = 0$, such that no additional occurrences whatsoever are required in $D\langle i\rangle$ since $C\langle j\rangle \vDash D\langle i\rangle$.

- ***Otherwise***, when the number of sub-trees matching C_j in the partial document sub-tree surpasses C_j's (maximum) cardinality constraints, i.e., $NbOcc[j] > R(C_j).MaxOccurs$, then we simply apply the *Traditional TED Recurrence* factor (γ_3), such that every additional C_j occurrence is treated as any regular sub-tree occurrence in the partial document sub-tree.

Since the three cases above express all possible tree edit paths following the set of edit operations considered in our approach, consequently we keep in Dist[*i*, *j*] the minimum from the three costs α, β, and γ, i.e., the minimum attainable distance between $D\langle i\rangle$ and $C\langle j\rangle$ allowing to transform $D\langle i\rangle$ into $D\langle i\rangle'$ such that $C\langle j\rangle \vDash D\langle i\rangle'$ □

4.2.3. Integrating TED⁺ in the Main Algorithm

When utilized in our main $TED_{XDoc\text{-}XGram}$ algorithm (Fig. 12), the *Extended TED Recurrence* (TED^+) is recursively called for every pair of sub-trees D_i and C_j in the document and grammar trees being compared (Fig. 12, lines 11-24). Here, the number of occurrences of document sub-trees evaluated as potential exact/approximate match candidates of grammar sub-tree $C_j \in C$, noted *NbOcc*[*j*], is compared with corresponding $R(C_j).MinOccurs$ and $R(C_j).MaxOccurs$ constraints (lines 15 and 17) in order to decide on the tree edit distance recurrence to execute (γ_1, γ_2, or γ_3). Then, the minimum edit distance between partial document tree $D\langle i\rangle$ and partial grammar tree $C\langle j\rangle$, highlighting the minimum cost scrip necessary to transform $D\langle i\rangle$ into $D\langle i\rangle'$ to obtain $C\langle j\rangle \vDash D\langle i\rangle'$, is kept in the distance matrix *Dist*[*i*, *j*].

Counter *NbOcc*[*j*], keeping track of the number of occurrences of document sub-trees D_i matching each grammar sub-tree C_j, is incremented when processing every D_i initially considered as a potential new (exact/approximate) match candidate for C_j (line 12). Then, *NbOcc*[*j*]'s new value is preserved whenever the edit distance cost $Dist[D_i, C_j] = Min(\alpha, \beta, \gamma) = \gamma$, i.e., whenever the cost of the edit scrip leading to $TED_{XDoc\_XGram}(D_i, C_j)$ (applying the γ factor), is lesser than those of: i) deleting D_i (applying the α factor), and ii) inserting C_j (applying the β factor). This means that the cheapest cost for transforming partial tree $D\langle i\rangle$ in order to obtain $C\langle j\rangle \vDash D\langle i\rangle'$, is through computing $TED_{XDoc\_XGram}(D_i, C_j)$ (i.e., through the γ factor), rather than deleting D_i or inserting C_j (applying the α or β factors), which in turn means (following the logic of edit distance) that D_i and C_j match; in other words that the potential match

candidate D_i is actually a confirmed match for C_j (either an exact match $C_j \vDash D_i$, when $\gamma = 0$, or an approximate match $C_j \mid\approx D_i$, when $\gamma \neq 0$). Otherwise, when the minimum distance $Dist[D_i, C_j] = Min(\alpha, \beta, \gamma) \neq \gamma$, then the $NbOcc[j]$ is reinitialized (line 25), hence ignoring D_i as a potential match for C_j.

At the end, the algorithm returns the minimum distance between all sub-trees (first-level, second-level, and so on) of the document tree D and grammar tree C (line 28), reflecting the minimum cost scrip necessary to transform D into D' to obtain $C \vDash D'$. The minimum distant value is then used to compute XML document/grammar similarity.

4.3. Similarity Measure

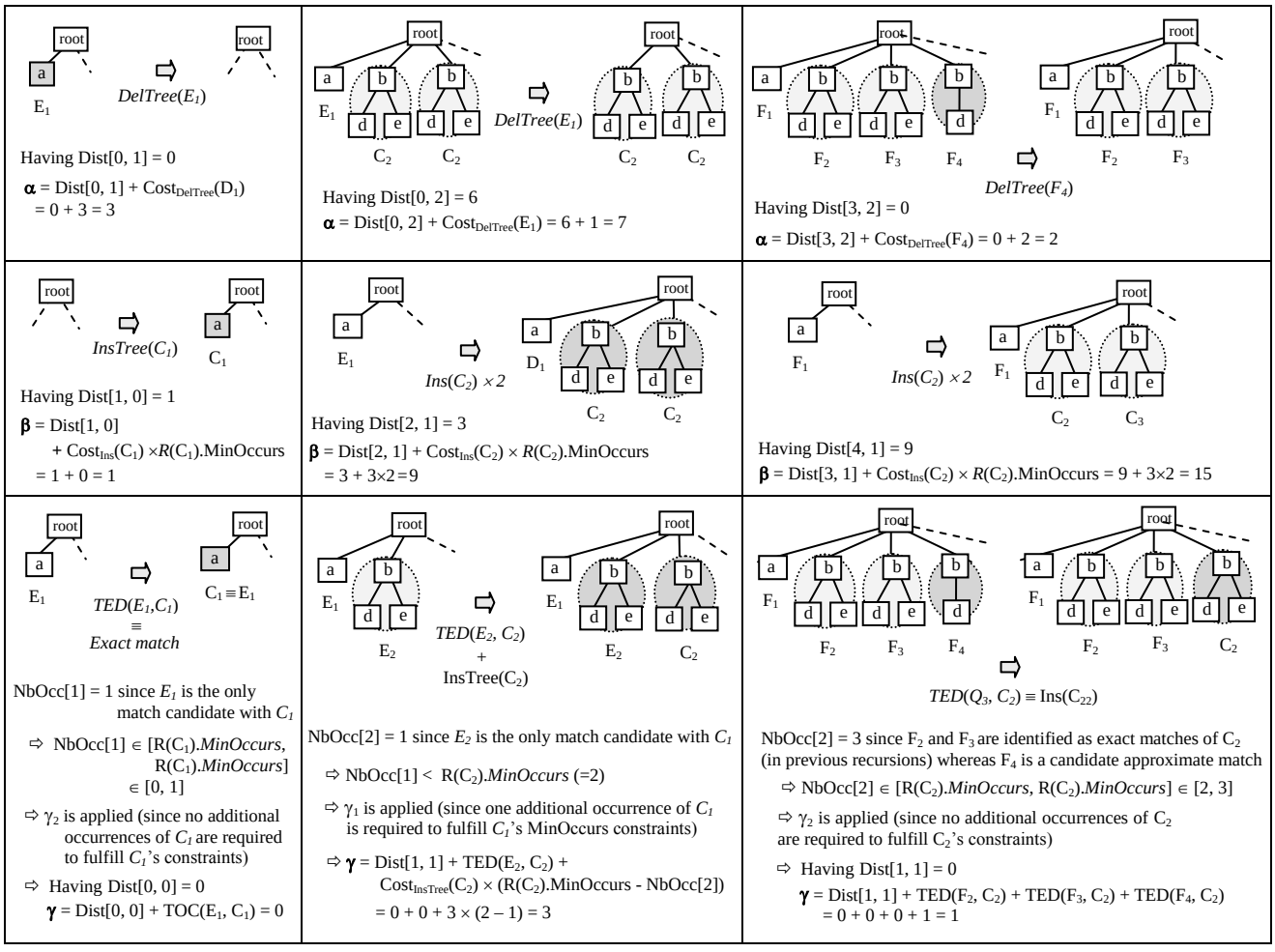
As indicated previously, we adopt the concept of *similarity* as the inverse of a *distance function* (a smaller distance value underlining a higher similarity degree). This minimal distance is computed using our TED_{XDoc_XGram} algorithm, such that our document/grammar similarity measure is defined as follows:

$$\mathrm{Sim}_{\mathrm{XDoc\_XGram}}(\mathrm{D}, \mathrm{C}) = \frac{1}{1 + \mathrm{TED}_{\mathrm{XDoc\_XGram}}(\mathrm{D}, \mathrm{C})} \quad \in [0, 1] \qquad \textbf{(4)}$$

When the XML grammar is represented as a set of conjunctive grammar trees $G = \{C\}_G$, the maximum similarity (i.e., minimum edit distance) between the XML document tree and the set of conjunctive grammar trees is retained:

$$\mathrm{Sim}_{\mathrm{XDoc\_XGram}}(\mathrm{D}, \mathrm{G}) = \underset{\mathrm{C_i} \in \{\mathrm{C}\}_\mathrm{G}}{Max} \left\{ \frac{1}{1 + \mathrm{TED}_{\mathrm{XDoc\_XGram}}(\mathrm{D}, \mathrm{C_i})} \right\} \quad \in [0, 1] \qquad \textbf{(5)}$$

Our similarity measure in formula (5) is consistent with the formal definition of similarity, as a (semi-) metric function satisfying (in part) the metric properties of *Reflexivity*, *Minimality*, *Symmetricity* and *Triangular Inequality* (cf. details in Appendix IV). Our measure is a semi-metric (and not a *full* metric) since: i) it does not allow comparing two grammars (i.e., $Sim(G_1, G_2)$), nor ii) using a grammar as the first parameter of the similarity measure ($Sim(G, D)$ is not allowed, i.e., we cannot transform grammar G in order to obtain $G' \vDash D$. We do it the other way around: transforming D to obtain $G \vDash D'$). Comparing/transforming grammars is out of the scope of this study.



a. Dist[1, 1] between *E* and *C*. **b.** Computing Dist[1, 2] between *E* and *C*. **c.** Computing Dist[4, 2] between *F* and *C*.

Fig. 14. Sample *Extended TED Recurrence* (TED^+) computations (to simplify, we note $TED_{XDoc\_XGram}(A, B)$ as $TED(A, B)$).

4.4. Computation Examples

4.4.1. TED$^+$ Computations

Consider the edit distance computations in Fig. 14. Fig. 14.a depicts the computation of *Dist*[1, 1] between partial document tree *E<1>* and partial grammar tree *C<1>*. Computing the α factor consists of computing the cost of deleting sub-tree E_1 (consisting of leaf node *a*), i.e., cost =1. Computing the β factor consists in inserting sub-tree C_1 (made of grammar node *a*) with $R(C_1).MinOccurs$ = 0, hence its cost = 0, indicating that C_1 is optional and is not required to appear in the partial document tree *D<1>* since *C<1>* $\vDash$ *D<1>*. Computing the γ factor consists in evaluating the edit distance between document sub-tree E_1, the (only existing) match candidate with grammar sub-tree C_2. Since $NbOcc[2]=1 \in [R(C_1).MinOccurs=0, R(C_1).MaxOccurs=1]$, thus γ_2 is applied. This yields cost = 0, indicating that E_1 is an exact match of C_1, $C_1 \vDash E_1$. Hence, $Dist[1, 1] = Min(\alpha, \beta, \gamma) = \gamma = 0$, indicating that no changes need to be made to *E<1>* since *C<1>* $\vDash$ *E<1>*.

Similar examples in Fig. 14.b and c are discussed in detail in Appendix V. To summarize, the example in Fig. 14.b depicts the computation of *Dist*[1, 2] between partial document tree *E<1>* and *C<2>*, where $Dist[1, 2] = Min(\alpha, \beta, \gamma) = \gamma = 3$, indicating that the minimum (cost) amount of change required to transform *E<1>* is to insert an additional occurrence of C_2 in *E<1>*, in order to obtain *C<2>* $\vDash$ *E<1>'*. The example in Fig. 14.c depicts the computation of *Dist*[4, 2] between partial document tree *F<4>* and partial grammar tree *C<2>*, where $Dist[4, 2] = Min(\alpha, \beta, \gamma) = \gamma = 1$, indicating that the minimum (cost) amount of change required to transform *F<4>* is to insert node *e* under sub-tree F_4, in order to obtain *C<2>* $\vDash$ *F<4>'*.

4.4.2. Complete TED_{XDoc_XGram} Matrix Computations

Fig. 15 shows the complete edit distance matrixes (with all recurrences) when running our TED_{XDoc_XGram} algorithm to compare document trees *D*, *E*, *F* with grammar *C* of Fig. 13.

For instance, the first line of the distance matrix in Fig. 15.a (likewise in Fig. 15.b and c), i.e., *Dist*[0][], corresponds to the sum of the costs of inserting every node of the grammar tree C_1. Likewise, the first column, *Dist*[][0], underlines the sum of the costs of deleting every node of XML tree *D*. Consequently, the algorithm identifies the combination of tree insertion/deletion operations of minimum cost, following our *Extended TED Recurrence*, in populating the remainder of the matrix, such as $TED_{XDoc\_XGram}(D, C)=Dist[|S|][|C|]$ underlines the final distance value.

The matrix in Fig. 15.a shows the edit distance result when comparing document tree *D* to grammar tree *C*, yielding $TED_{XDoc\_XGram}(D, C) = 0 \Rightarrow Sim_{XDoc\_XGram}(D, C) = 1 / (1+TED_{XDoc\_XGram}(D, C)) = 1 \Rightarrow C \vDash D$. The minimum cost edit script is highlighted in Fig. 15.a. $Dist[0, 0] = Cost_{Upd}(R(D), R(C))= 0$, since the document/grammar tree roots match: $R(D) = R(C)$. $Dist[0, 1] = Dist[0, 0] + Cost_{InsTree}(C_1) \times R(C_1).MinOccurs = 0$, underlining that C_1 is optional and is not required to appear in the document tree. $Dist[3, 2] = Dist[0, 1] + TED_{XDoc\_XGram}(D_1, C_2) + TED_{XDoc\_XGram}(D_2, C_2) + TED_{XDoc\_XGram}(D_3, C_2) = 0$ since $C_2 \vDash \{D_1, D_2, D_3\}$, such that $NbOcc[2] = R(C_3).MaxOccurs = 3$ (3 occurrences of C_2 are allowed to appear, and have actually appeared in the document tree). $Dist[4, 3] = Dist[3, 2] + TED_{XDoc\_XGram}(D_4, C_3) \equiv Cost_{Upd}(R(D_4).\ell, R(C_3).\ell) = 0$ since $C_3 \vDash D_4$, having $R(C_3).MinOccurs = R(C_3).MaxOccurs = 1$ (i.e., one and only one occurrence of C_3 is required to appear in the document tree). Hence, no changes need to be made to *D* since $C \vDash D$.

Similar examples in Fig. 15.b and c show the edit distance result when comparing document trees *E* and *F* (respectively) with grammar tree *C*, and are discussed in detail in Appendix V. To summarize Fig. 15.b shows $TED_{XDoc\_XGram}(E, C) = 3 \Rightarrow Sim_{XDoc\_XGram}(E, C) = 1 / (1 + TED_{XDoc\_XGram}(E, C) = 0.25 \Rightarrow C \mid\approx_{0.25} E$ highlighting the cost of inserting one additional occurrence of sub-tree C_2 into document tree *E*, to obtain $C \vDash E'$. Similarly, Fig. 15.a shows $TED_{XDoc\_XGram}(F, C) = 4 \Rightarrow Sim_{XDoc\_XGram}(F, C) = 1 / (1 + TED_{XDoc\_XGram}(F, C) = 0.2 \Rightarrow C \mid\approx_{0.2} F$, which underlines the costs of i) inserting node *e* in sub-tree F_4, and ii) deleting sub-tree F_5 from document tree *F*, in order to obtain $C \vDash F'$. This means that *F* requires more costly transformations to become valid w.r.t. grammar tree *C*, and thus is less similar to grammar tree *C* in comparison with document tree *E*.

4.4.3. Running Example

To sum up, we present the result of comparing sample XML document *Paper.xml* with XML grammar *Paper.xsd* in Fig. 16 (reported from Fig. 2.a and Fig. 7.c respectively, for ease of presentation). *Paper.xml* is represented as XML document tree *D* following our XML data tree model (Fig. 16.a), and *Paper.xsd*, designated as *G*, is represented as a set of conjunctive grammar trees $\{C_I, C_{II}, C_{III}\}$. TED_{XDoc_XGram} computations between *D* and $\{C_I, C_{II}, C_{III}\}$ yield:

- $TED_{XDoc\_XGram}(D, C_I) = TED_{XDoc\_XGram}(D_2, C_{I_3}) + Cost_{InsTree}(C_{I_3}) \times (R(C_{I_3}).MinOccurs - NbOcc[3]) = 1 + 3 = 4$, which comes down to: i) the cost of transforming D_2, as an approximate match candidate of sub-tree C_{I_3}, in order to obtain $C_{I_3} \vDash D_2'$ ($\equiv Cost_{Upd}(R(D_2).\ell, R(C_{I_3}).\ell)=1$, updating node label *Publisher* into *Author*), and ii) the cost of inserting one additional occurrence of C_{I_3} (made of nodes *Author*, *FirstName* and *LastName*, i.e., $Cost_{InsTree}(C_{I_3}) = 3$), since D_2 is the only match of C_{I_3} in *D* (*NbOcc*[2] = 1) whereas the minimum number of occurrences of C_{I_3} required to appear in document tree *D* is $R(C_{I_3}).MinOccurs = 2$, transforming *D* into *D'* in order to obtain $C_I \vDash D'$.

- $TED_{XDoc\_XGram}(D, C_{II}) = \text{Cost}_{\text{DelTree}}(D_{21}) + \text{Cost}_{\text{DelTree}}(D_{22}) = 2$, which corresponds to the sum of the costs of deleting (sub-tree) nodes of labels *FirstName* and *LastName* from document tree *D*, to obtain $C_{II} \models D'$.
- $TED_{XDoc\_XGram}(D, C_{III}) = 0$, underlining that changes need to be made to document *D*, since we already have $C_{III} \models D$ (edit distance matrixes when comparing *D* with C_I, C_{II}, and C_{III} can be found in Appendix V).

i ↓ \ *j* →	*0*	*1*	*2*	*3*
	R(C) $^{1}_{1}$	$C_1{}^{1}_{0}$	$C_2{}^{\infty}_{2}$	$C_3{}^{1}_{1}$
0 R(D)	0	0	6	7
1 D_1	3	$min\begin{pmatrix}\alpha=3\\ \beta=3\\ \gamma=3\end{pmatrix}=3$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=9\\ \beta=9\\ \gamma=3\end{pmatrix}=3$ NbOcc[2]=1	$min\begin{pmatrix}\alpha=10\\ \beta=1\\ \gamma=9\end{pmatrix}=1$ NbOcc[3]=0
2 D_2	6	$min\begin{pmatrix}\alpha=6\\ \beta=6\\ \gamma=6\end{pmatrix}=6$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=12\\ \beta=12\\ \gamma=0\end{pmatrix}=0$ NbOcc[2]=2	$min\begin{pmatrix}\alpha=4\\ \beta=1\\ \gamma=3\end{pmatrix}=1$ NbOcc[3]=0
3 D_3	9	$min\begin{pmatrix}\alpha=9\\ \beta=9\\ \gamma=9\end{pmatrix}=9$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=9\\ \beta=9\\ \gamma=0\end{pmatrix}=0$ NbOcc[2]=3	$min\begin{pmatrix}\alpha=7\\ \beta=1\\ \gamma=3\end{pmatrix}=1$ NbOcc[3]=0
4 D_4	10	$min\begin{pmatrix}\alpha=10\\ \beta=10\\ \gamma=10\end{pmatrix}=10$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=1\\ \beta=16\\ \gamma=10\end{pmatrix}=1$ NbOcc[2]=0	$min\begin{pmatrix}\alpha=8\\ \beta=2\\ \gamma=0\end{pmatrix}=0$ NbOcc[3]=1

a. Comparing document tree *D* and grammar tree C.

i ↓ \ *j* →	*0*	*1*	*2*	*3*
	R(C) $^{1}_{1}$	$C_1{}^{1}_{0}$	$C_2{}^{3}_{2}$	$C_3{}^{1}_{1}$
0 R(D)	0	0	6	7
1 E_1	1	$min\begin{pmatrix}\alpha=1\\ \beta=1\\ \gamma=0\end{pmatrix}=0$ NbOcc[1]=1	$min\begin{pmatrix}\alpha=7\\ \beta=6\\ \gamma=3\end{pmatrix}=3$ NbOcc[2]=1	$min\begin{pmatrix}\alpha=8\\ \beta=4\\ \gamma=7\end{pmatrix}=4$ NbOcc[3]=0
2 E_2	4	$min\begin{pmatrix}\alpha=3\\ \beta=4\\ \gamma=7\end{pmatrix}=3$ NbOcc[1]=1	$min\begin{pmatrix}\alpha=6\\ \beta=9\\ \gamma=3\end{pmatrix}=3$ NbOcc[2]=2	$min\begin{pmatrix}\alpha=7\\ \beta=4\\ \gamma=6\end{pmatrix}=4$ NbOcc[3]=0
3 E_3	5	$min\begin{pmatrix}\alpha=4\\ \beta=5\\ \gamma=5\end{pmatrix}=4$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=4\\ \beta=10\\ \gamma=4\end{pmatrix}=4$ NbOcc[2]=0	$min\begin{pmatrix}\alpha=5\\ \beta=5\\ \gamma=3\end{pmatrix}=3$ NbOcc[3]=1

b. Comparing document tree *E* and grammar tree *C*.

i ↓ \ *j* →	*0*	*1*	*2*	*3*
	R(C) $^{1}_{1}$	$C_1{}^{1}_{0}$	$C_2{}^{\infty}_{2}$	$C_3{}^{1}_{1}$
0 R(D)	0	0	6	7
F$_1$	1	$min\begin{pmatrix}\alpha=1\\ \beta=1\\ \gamma=0\end{pmatrix}=0$ NbOcc[1]=1	$min\begin{pmatrix}\alpha=9\\ \beta=6\\ \gamma=6\end{pmatrix}=6$ NbOcc[2]=0	$min\begin{pmatrix}\alpha=8\\ \beta=7\\ \gamma=7\end{pmatrix}=7$ NbOcc[3]=0
2 F$_2$	4	$min\begin{pmatrix}\alpha=3\\ \beta=4\\ \gamma=4\end{pmatrix}=3$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=12\\ \beta=12\\ \gamma=3\end{pmatrix}=3$ NbOcc[2]=1	$min\begin{pmatrix}\alpha=10\\ \beta=1\\ \gamma=9\end{pmatrix}=1$ NbOcc[3]=0
3 F$_3$	7	$min\begin{pmatrix}\alpha=6\\ \beta=7\\ \gamma=7\end{pmatrix}=7$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=3\\ \beta=13\\ \gamma=0\end{pmatrix}=0$ NbOcc[2]=2	$min\begin{pmatrix}\alpha=4\\ \beta=1\\ \gamma=3\end{pmatrix}=1$ NbOcc[3]=0
4 F$_4$	9	$min\begin{pmatrix}\alpha=9\\ \beta=9\\ \gamma=10\end{pmatrix}=9$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=2\\ \beta=15\\ \gamma=1\end{pmatrix}=1$ NbOcc[2]=3	$min\begin{pmatrix}\alpha=3\\ \beta=2\\ \gamma=2\end{pmatrix}=2$ NbOcc[3]=0
5 F$_5$	12	$min\begin{pmatrix}\alpha=12\\ \beta=12\\ \gamma=12\end{pmatrix}=12$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=4\\ \beta=18\\ \gamma=10\end{pmatrix}=4$ NbOcc[2]=0	$min\begin{pmatrix}\alpha=5\\ \beta=5\\ \gamma=4\end{pmatrix}=4$ NbOcc[3]=1
6 F$_6$	13	$min\begin{pmatrix}\alpha=13\\ \beta=13\\ \gamma=13\end{pmatrix}=13$ NbOcc[1]=0	$min\begin{pmatrix}\alpha=5\\ \beta=19\\ \gamma=15\end{pmatrix}=5$ NbOcc[2]=0	$min\begin{pmatrix}\alpha=5\\ \beta=6\\ \gamma=4\end{pmatrix}=4$ NbOcc[3]=0

c. Comparing document tree *F* and grammar tree *C*.

Fig. 15. Computing edit distance between XML documents *D*, *E* and *F* and grammar tree *C* in Fig. 13.

Hence, the structural similarity between XML document *Paper.xml* and XML grammar *Paper.xsd* is computed as:

$$\text{Sim}_{\text{XDoc\_XGram}}(D,G) = \underset{C_i \in \{C_{II}, C_{II}, C_{III}\}}{Max}\left\{\frac{1}{1+\text{TED}_{\text{XDoc\_XGram}}(D,C_i)}\right\} = \frac{1}{1+\text{TED}_{\text{XDoc\_XGram}}(D,C_{III})} = 1 \quad \Rightarrow \quad G \models D$$

In other words, a maximum similarity value (1 or 100%), indicates that XML document *Paper.xml* is structurally valid w.r.t. grammar *Paper.xsd*, and that no transformations need to be applied to the corresponding document tree since it already conforms to the grammar tree representation.

4.5. Complexity Analysis

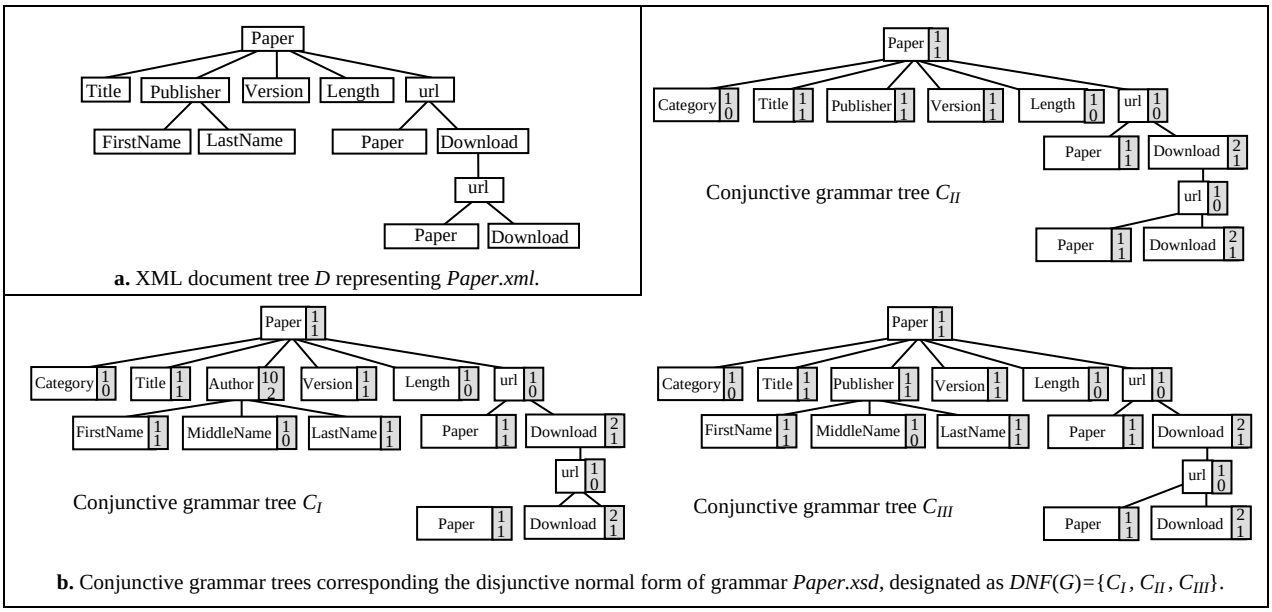
Let $|D|$ be the cardinality of the XML document tree *D* considered, and $|G|$ the number of nodes (elements/attributes) in the XML grammar, $N_G = |\{C\}_G|$ the number of conjunctive grammars making up the *disjunctive normal form* of *G*, and $|C_G|$ the cardinality of the largest conjunctive grammar tree corresponding to *G*. Our XML document and grammar structure comparison approach is of $O(|D|+|G|+(N_G \times |D| \times |C_G|))$ time. It simplifies to $O(|D| \times |G|)$ in the typical (practical) case, and $O(N_G \times |D| \times |G|)$ in the worst case.

4.5.1. Time Complexity

Tree Construction: The XML document tree and XML grammar tree construction processes (including algorithm *XGram_to_Tree*) are of typical linear complexity and simplify to $O(|D|+|G|)$. Algorithm *XGram_to_Tree* processes XML grammar simplification rules using a dedicated index tables to monitor each simplification rule (i.e., detecting whether the grammar expression is of the form targeted by a given transformation rule). This proved computationally efficient in practice, requiring typical $O(|G|)$, since the number of simplification rules – and thus the size of the index tables – is constant). On the other hand, document tree construction requires one single traversal over the document, hence $O(|D|)$ time.

Tree Edit Operations Costs: Computing document tree and conjunctive grammar tree edit operations' costs requires $O(|D|+|C_G|)$ time: i) algorithm TOC_{XGram} for computing XML grammar tree edit operations costs is of $O(|C_G|)$ time, ii) Likewise, algorithm TOC_{XDoc} (developed in the Technical Report [73]) for computing document tree operations costs, requires $O(|D|)$ time.

Core Tree Edit Distance Algorithm: The TED_{XDoc_XGram} algorithm (Fig. 12) for computing the edit distance between the XML document tree and conjunctive grammar tree is of worst $O(|D|\times|C_G|)$ complexity. The algorithm recursively goes through the sub-trees of both XML document and conjunctive grammar trees, combining edit operations so as to identify those of minimal cost. Its main recursive procedure is called once for each pair of sub-trees occurring and the same structural level (depth) in the document and conjunctive grammar trees being compared, thus reflecting a linear dependency on the size of each tree, and thus a quadratic dependency on the sizes of both trees.



a. XML document tree *D* representing *Paper.xml.*

b. Conjunctive grammar trees corresponding the disjunctive normal form of grammar *Paper.xsd*, designated as $DNF(G)=\{C_I, C_{II}, C_{III}\}$.

Fig. 16. XML document tree (reported from Fig. 2.a) and conjunctive grammar trees (reported from Fig. 7.c).

XML Document/Grammar Comparison: Algorithms TOC_{XGram} and TED_{XDoc_XGram} are executed for all conjunctive grammars $C_i \in \{C\}_G$ in order to compute overall document/grammar edit distance similarity, thus requiring $O(N_G\times|D|\times|C_G|)$ time. Here, recall that the number of conjunctive grammars N_G resulting from the *disjunctive normal form* expansion of an input XML grammar *G*, depends on the number and configurations of *Or* (choice) operators in the input grammar expressions. This may generate a proliferation of conjunctive grammars depending on the expressiveness of the grammar declarations. However, in our approach, the XML document tree and each of the conjunctive grammar trees are compared concurrently (i.e., in parallel) using multi-thread processing (cf. algorithm in Fig. 9). Hence, regardless of the (possibly limited) processing capabilities of the computer system being used, the complexity of the edit distance phase is not (theoretically) affected by the number of conjunctive grammars N_G, and comes down to $O(|D|\times|C_G|)$. In addition, most common *alternative* expressions found in real XML grammars [11, 23, 38] generate a number of conjunctive grammars linear in the number of *Or* operators involved (cf. mathematical analysis is Appendix I). Hence, based on i) the algorithmic design of our approach (Fig. 9), and i) the relatively simple nature of real XML grammar expressions, the overall complexity of our approach, $O(N_G\times|S|\times|C_G|)$, typically simplifies to $O(D|\times|C_G|)$, which in turn simplifies to $O(|D|\times|G|)$, since $|C_G|$ is linear in the size of $|G|$ (cf. Section 3.2).

In the worst case, when the number of conjunctive grammars N_G is explosive (and cannot be even handled using multi-threading), then the term N_G cannot be simplified form the equation, and complexity becomes $O(N_G\times|D|\times|G|)$.

4.5.2. Space Complexity

As for memory consumption, our approach requires $O(|D| + N_G\times|C_G|)$ to store the XML document tree and conjunctive XML grammar trees being compared, in addition to $O(N_G\times|D|\times|C_G|)$ space to store corresponding distance matrixes. Yet practically, space complexity simplifies to $O(|D|+|G|) + O(|D|\times|G|) = O(|D|\times|G|)$ since conjunctive grammar trees consist of references (pointers) to the elements/attributes in the source grammar, and thus require limited

space in comparison with the actual document and grammar sizes (even when the child structures of elements in different conjunctive grammar trees are different, represented by their respective pointers). Experimental time and space analyses are provided in Section 5.7.

5. Experimental Evaluation

We first start by describing our prototype and experimental scenarios, and then we present and assess empirical results.

5.1. Prototype

We have implemented our XML document/grammar comparison approach in the existing *XS3* prototype[1] (*XML Structural and Semantic Similarity*). Implemented using C#.Net, the *XS3* prototype system includes: i) a *parser component* verifying the integrity of XML documents and grammars, ii) a *tree representation component*, for transforming XML documents and grammars into their tree representations, and iii) a *tree edit distance component* for computing document/grammar similarity. An adaptation of the IBM XML documents generator[2] was implemented to produce sets of XML documents and grammars based on specific user input requirements (e.g., a *MaxRepeats*[3] variability parameter for document generation, the number of *'And/Or'* operators and operator positions in synthetic grammars, etc.). In addition, we have implemented an XML document/grammar modification generator. It accepts as input an XML document or an XML grammar, a *ModifType* value designating the kind of modification to be induced to the document/grammar at hand (i.e., element/attribute *insertions*, *deletions* or *label updates*, cf. Section 5.4), as well as a *Modif%* value indicating the amount of modifications to be produced w.r.t. document/grammar size (i.e., cardinality).

Built upon the main *XS3* components are different modules for similarity evaluation: *One to One*, *One to Many* (comparing one XML document to a set of grammars and vice-versa, allowing similarity ranking), and *Set comparison*, (enabling XML document/grammar classification). A detailed description of the prototype system is available online.

5.2. Experimental Scenarios

How to experimentally evaluate the quality of an XML similarity method remains a debatable issue, especially in information retrieval. To our knowledge, the definition of standardized XML similarity evaluation metrics remains a hot topic in the INEX evaluation campaigns[4]. While a few similarity evaluation techniques have been proposed in the context of XML document comparison (e.g., *inter-* and *intra-cluster similarity* coefficients [30], *mis-clustering* coefficient [48], and *cluster-precision* and *-recall* metrics [26]), and grammar comparison (e.g., *overall* measure to quantify user effort in grammar matching [27, 44]) yet, to our knowledge, none have been proposed for XML document/grammar similarity evaluation; which is probably due to the novelty of the issue.

Hence, in the following, we introduce experimental evaluation methods based on the most common applications of XML document/grammar comparison, i.e., document classification and ranked retrieval. We demonstrate our method's effectiveness in classifying similar documents w.r.t. predefined grammars in Section 5.3, and ranking relevant XML documents (grammars) w.r.t. their resemblances to the grammars (documents), in Section 5.4. In Section 5.5, we perform a hybrid experimental analysis, combining both document classification and grammar transformation, to assess our method's 'intelligent' (noise resistant) behavior in comparing non-conforming yet related documents/grammars, i.e., given a set of grammars, recognizing documents which are similar but are not written exactly in those grammars. A qualitative comparative study is presented in Section 5.6. Complexity analysis is presented in Section 5.7.

5.3. XML Document Classification Experiments

The scenario adopted in our document classification experiments comprises of 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 local grammars. Consequently, XML documents introduced in a given database are matched, via an XML structural similarity method, against the local grammars. In such an application, a similarity threshold is identified underlining the minimal degree of similarity required to bind an XML document to a grammar. The XML grammar for which the similarity degree is highest, and above the specified threshold, is selected. Thus, the XML document is accepted as approximately valid for that grammar (the documents are exactly valid when similarity is maximal, i.e., $Sim_{XDoc\_XGram}=1$).

Note that when the similarity score is below the threshold, for all grammars in the XML database, the XML document is considered unclassified and is stored separately.

5.3.1. Evaluation Metrics

Owing to the proficient use of their traditional predecessors in classic information retrieval evaluation [43], and their recent exploitation in XML document clustering (e.g., [26]), we adapt the *precision* metric (*PR*, highlighting the fraction of relevant selected entities) and the *recall* metric (*R*, highlighting the fraction of relevant non-selected entities)

---

[1] Available online at http://sigappfr.acm.org/Projects/XS3/

[2] http://www.alphaworks.ibm.com.

[3] A greater *MaxRepeats* underlines greater size and variability in generating XML documents, when repeatable elements (associated *, + in DTDs, or *MaxOccurs* in XSDs) are encountered.

[4] http://inex.is.informatik.uni-duisburg.de/

in information retrieval to our XML classification scenario, and propose a new method for their usage in order to obtain consistent experimental results. For an extracted class K_i corresponding to a given grammar G_i:

- a_i is the number of XML documents in K_i that indeed correspond to G_i (correctly classified documents, i.e., those that conform to grammar G_i).
- b_i is the number of documents in K_i that do not correspond to G_i (misclassified).
- c_i is the number of XML documents not in K_i, although they correspond to G_i (documents that conform to G_i and that should have been classified in K_i).

Hence, setting n as the total number of classes, which corresponds to the total number of grammars considered for the classification task, we have:

$$PR = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i}, \qquad R = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} c_i}, \qquad F\text{-}Value = \frac{2 \times PR \times R}{PR + R} \tag{6}$$

High *precision* denotes that the classification task achieved high accuracy grouping together documents that actually correspond to the grammars considered. On the other hand, high *recall* means that very few XML documents are not in the appropriate class where they should have been. In addition to comparing one approach's *precision* improvement to another's *recall*, it is also a common practice to consider their harmonic mean: the *F-value* measure. Hence, as with classic information retrieval, high *precision* and *recall*, and thus high *F-value*, (indicating in our case high classification quality) characterize a good (XML document/grammar) similarity method.

5.3.2. Multi-level Classification

In our experiments, we undertook a series of multilevel classification tasks, varying the classification threshold in the *[0, 1]* interval. In other words, we construct a dendrogram-like structure (Fig. 17.a) such that:

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

Then, we compute *precision*, *recall* and *F-value* for each classification set identified in the dendrogram, thus constructing *PR*, *R* and *F-value* graphs that describe the system's evolution throughout the classification process.

5.3.3. Experimental Results

We conducted experiments on both real and synthetic XML documents to test our XML document/grammar structural comparison method. For real XML data, we utilized the online XML version of the ACM SIGMOD Record[1], and the University of Wisconsin's Niagara XML document collection[2], including a large XML data set extracted from the Internet Movie Database IMDB[3]. We performed two main classification experiments to test the effectiveness of our method in comparing: i) related XML documents (i.e., documents sharing identical tag names and related structures), and ii) heterogeneous XML documents (describing different kinds of information, using different structures). The first experiment considers the SIGMOD Record documents, which correspond to three main grammars: *OrdinaryIssuePage.dtd*, *ProceedingsPage.dtd* and *SigmodRecord.dtd* [4], describing scientific publications. The second experiment considers all three SIGMOD Record, Niagara and IMDB data sets, combining heterogeneous XML data describing different kinds of information, ranging over scientific publications, company profiles, personnel descriptions, movie credentials, and actor descriptions. The characteristics of each document collection and corresponding grammar definitions are shown in Table 3. Grammar statistics are shown in Table 5.

We also generated two sets of 1000 XML documents from 20 real-case[5] and synthetic grammars (using the synthetic XML document and XML grammar generators implemented in the *XS3* prototype). The first set of documents was created with *MaxRepeats* = 5, the second with *MaxRepeats* = 10, the latter set underlining XML documents with greater size and variability (i.e., greater heterogeneity) w.r.t. the former, when optional and repeatable elements are encountered. The characteristics of synthetic XML datasets are summarized in Table 4 and Table 5.

---

[1] Available at http://www.acm.org/sigmod/xml

[2] Available at http://www.cs.wisc.edu/niagara/

[3] XML data extracted from http://www.imdb.com/ using a dedicated wrapper generator[2].

[4] We were able to find only one XML file conforming to *SigmodRecord.dtd*: *SigmodRecord.xml*. However, due to its relatively large size (479KB) w.r.t. the XML documents corresponding to the other grammars (12KB of average size per document), we carefully decomposed *SigmodRecord.xml* to several documents, creating a set of documents conforming to *SigmodRecord.dtd*.

[5] From http://www.xmlfiles.com and http://www.w3schools.com.

Table 3. Characteristics of SIGMOD Record, Niagara, and IMBD document sets.

Grammars[1] (SIGMOD)	**N# of Docs**	**Avg Node Depth** (per doc)	**N# of nodes** (per gram)	**Avg N# of nodes** (per doc)
OrdinaryIssuePage	30	5.49	23	262.81
ProceedingsPage	47	3.67	31	382.72
SigmodRecord	27	5.77	14	542.92

Grammars (Niagara)	**N# of Docs**	**Avg Node Depth** (per doc)	**N# of nodes** (per gram)	**Avg N# of nodes** (per doc)
Profile	141	2.57	12	381.25
Personnel	20	2.63	14	38.35
Club	11	2.19	13	259.09
Bib	15	3.04	14	130.66

Grammars (IMDB)	**N# of Docs**	**Avg Node Depth** (per doc)	**N# of nodes** (per gram)	**Avg N# of nodes** (per doc)
Movies	300	3.31	12	53.71
Actors	300	3.40	9	120.25

Table 4. Characteristics of synthetic document sets.

Document set	**N# of grammars**	**Number of Documents**	**N# of documents (per gram)**	**Average Node Depth (per doc)**	**Average Number of Nodes (per doc)**
MaxRepeats = 5	20	1000	50	3.1	15.4768
MaxRepeats = 10	20	1000	50	3.68	36.9133

Table 5. The percentage and number of structure model expressions in both sets of real and synthetic grammars.

Grammar set	**Sequence exp. (*And*)**	**Alternative exp. (*Or*)**	**Mixed exp. (*And* & *Or*)**	**Single element Expressions**	**Empty structural model (ε) exp.**
Real grammars	19.10 % (51)	1.12 % (3)	1.87 % (5)	14.98 % (40)	62.92 % (168)
Synthetic grammars	5.59 % (8)	6.99 % (10)	9.79 % (14)	10.48 % (15)	67.13 % (96)

Note that grammar statistics in Table 5 fairly concur with the empirical analyses in [11, 23, 38] highlighting the fact that real-world XML grammars are usually made of simple structural models (e.g., sequence expressions, single element declarations, or basic content models, e.g., *PCDATA*, *String*, etc.). In other words, few grammar expressions contain alternative declarations, i.e., *Or* operators (e.g., less than 7% of all grammar expressions surveyed in [11], and less than 16% of those surveyed in [23], contain *Or* operators - cf. 0 for preliminary statistics).

In addition, note that all real and synthetic grammars considered in our experiments are fairly different and do not produce identical documents. In other words, we made certain that a given document cannot conform to two grammars simultaneously, so as to prevent any confusion in computing the *precision* and *recall* metrics. *Precision* and *recall* graphs are presented in Fig. 17. One can clearly realize that *recall* (*R*) is always equal to 1. This reflects the fact that our XML document/grammar comparison approach constantly identifies, in the grammar classes, the XML documents that actually conform to the grammars considered (i.e., documents having Sim_{XDoc_XGram}=1), regardless of the classification threshold as well as the nature of the document collection (related and/or heterogeneous). On the other hand, *precision* (*PR*), and consequently *F-value* (note that *F-value* follows *PR* in this experiment, since *R* is always equal to 1) gradually increases toward 1, while varying the classification threshold from 0 to 1:

- When the classification threshold is equal to 0, all documents in the XML repository are considered in each and every class corresponding to the grammars at hand (Fig. 17.a, *initial level*). That is underlined by minimum *PR*.
- Then, as the classification threshold increases, inconsistent documents are gradually filtered from the XML grammar classes, ultimately yielding classes that only encompass documents conforming to the considered grammars (cf. Fig. 17.a, *final level*).

In summary, *Precision* and *F-value* results in Fig. 17 show that our method yields high classification quality with both related and heterogeneous document collections, obtaining optimal classes at a very early stage of the multilevel classification process (with thresholds < 0.5).

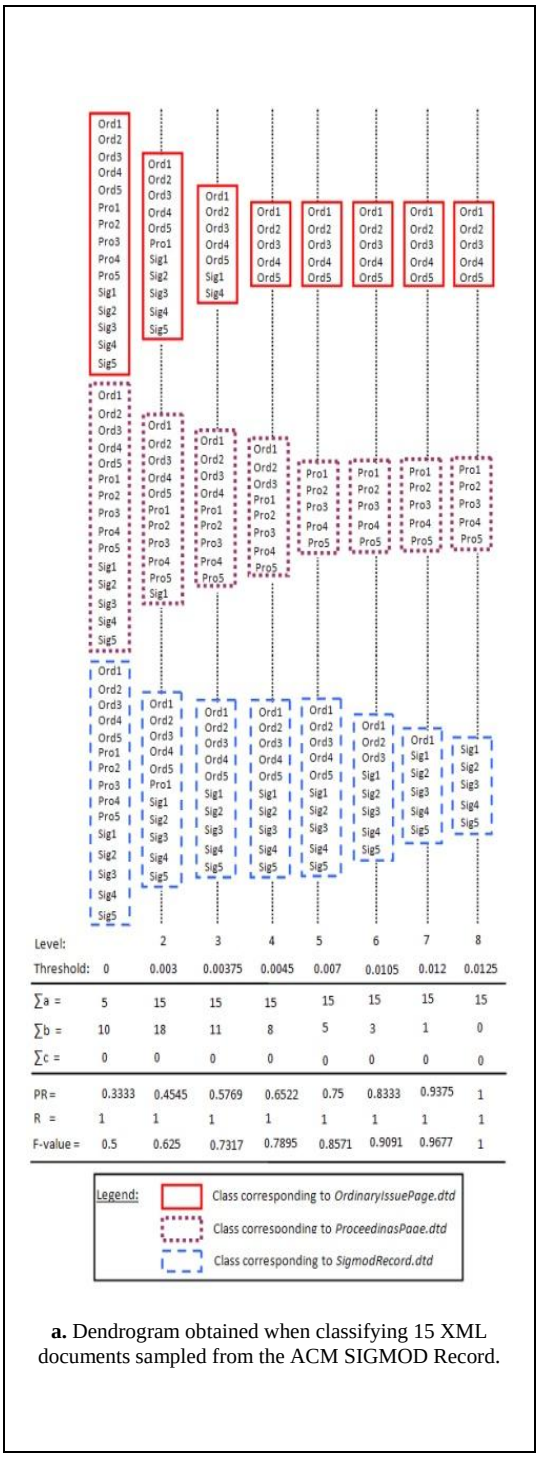
5.4. Similarity Ranking Experiments

In addition to XML document classification, we ran a series of experiments to evaluate the ranking capabilities of our document/grammar comparison method.

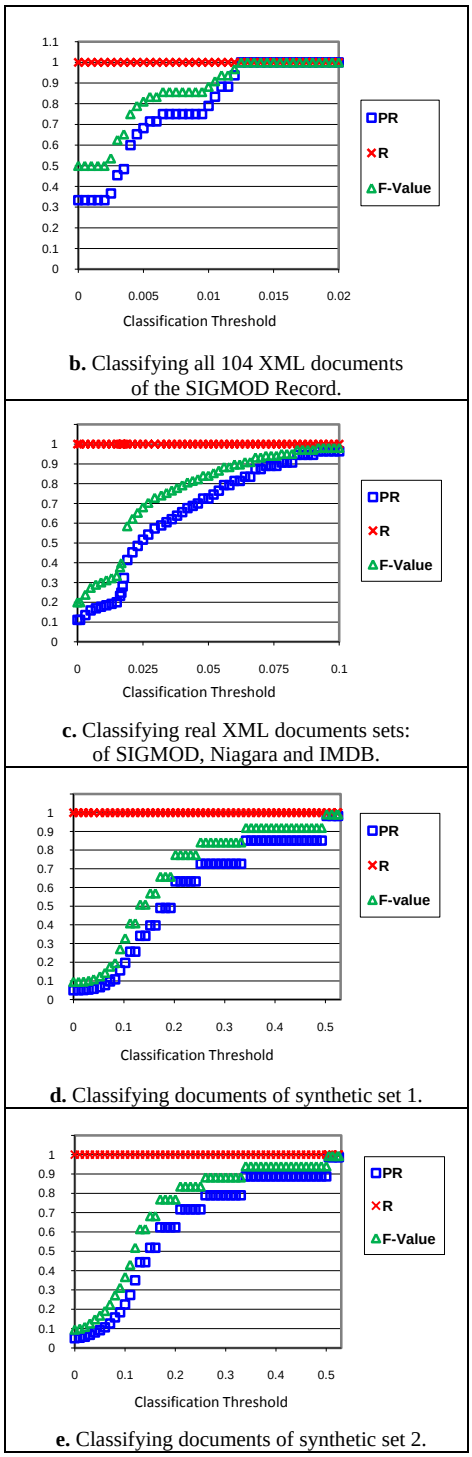
5.4.1. Experimental Scenario

The approach consists in gradually transforming real XML documents/grammars, and consequently evaluating how closely the obtained similarity results correspond to the induced changes. Here, we exploit two complementary criteria for ranking evaluation: i) an internal criterion, consisting of the amount of modification (transformation) in documents/grammars, and ii) an external criterion, consisting of user predefined rankings. On one hand, we consider as an internal evaluation criterion: the correspondence between the amount of changes and document/grammar similarity values (i.e., similarity decreasing proportionally with the increase in changes, and vice-versa), such that a straight correspondence would underline high ranking quality. On the other hand, we also exploit user-predefined rankings, necessary to highlight the user's perception of document/grammar similarity w.r.t. document/grammar modifications.

[1] Note that all DTD grammars were transformed into XSD definitions, replacing DTD cardinality constraints (namely: *?*, *, +) with their more expressive XSD counterparts (i.e., *MinOccurs* and *MaxOccurs*).



a. Dendrogram obtained when classifying 15 XML documents sampled from the ACM SIGMOD Record.



b. Classifying all 104 XML documents of the SIGMOD Record.

c. Classifying real XML documents sets: of SIGMOD, Niagara and IMDB.

d. Classifying documents of synthetic set 1.

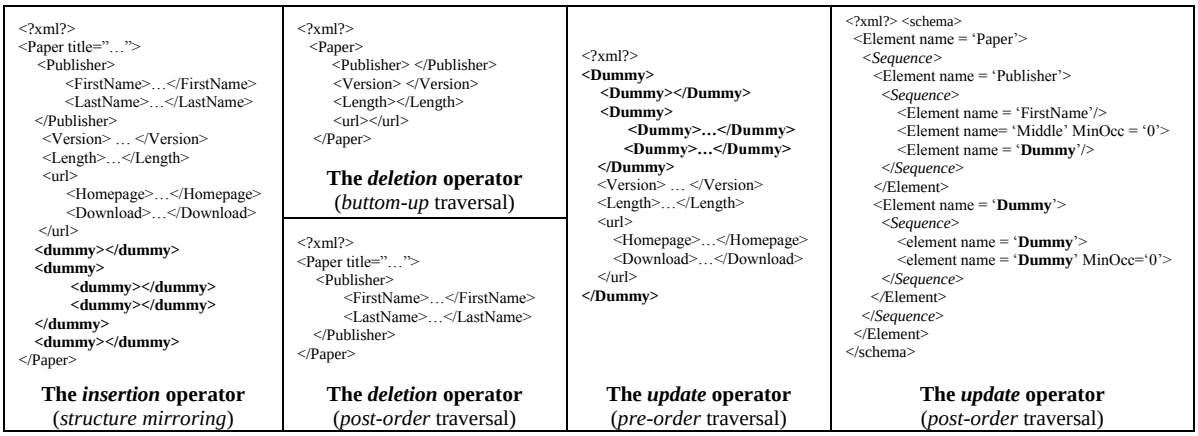
e. Classifying documents of synthetic set 2.

Fig. 17. XML document classification: dendrogram, and *PR*, *R*, *F-value* graphs.

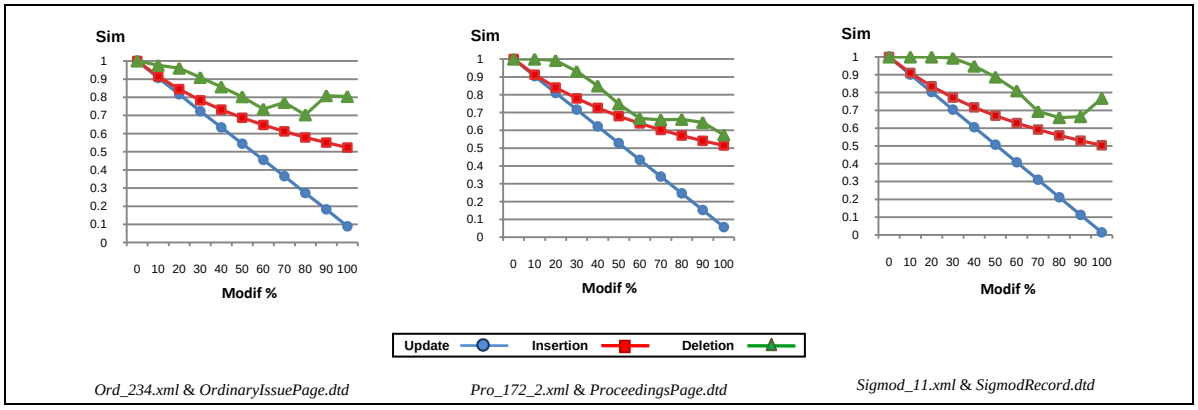
To produce changes to XML documents/grammars (cf. Fig. 18.a), we utilize our prototype's modification generator:

- For the starting phase of the transformation process, the modification threshold *Modif%* is set to 0, underlining the original document/grammar structure.
- For the final phase, *Modif%*=100. The amount of changes in the resulting modified document/grammar at hand amounts to *100%* of its original size.
- Intermediate transformation phases correspond to 0< *Modif%* <100.

In addition, for each similarity ranking experiment, the modified documents/grammars were manually evaluated, identifying corresponding user-relevant rankings. Thirty graduate students were involved in the experiments. Each subject was given a set of initially conforming documents/grammars and their transformed (modified) versions, and was asked to rank the transformed documents/grammars w.r.t. the original versions (assigning scores ranging from *A* to *F*, such as *A = Conforming*, *B = Very Similar*,…, *F = Least similar*). Manual answers were consequently correlated against the system generated ones in order to identify the statistical dependence between system generated similarity scores and the user's perception of similarity.



a. Samples of utilizing the modification operators, at *Modif%* = 50 (i.e., modifying 50% of original document).



b. Similarity graphs reflecting the gradual modification of XML documents *Ord_234.xml*, *Pro_171_2.xml* and *Sigmod_11.xml*, w.r.t. XML grammars *OrdinaryIssuePage.dtd*, *ProceedingsPage.dtd* and *SigmodRecord.dtd*.

Fig. 18. Detecting and measuring changes (modifications) in XML documents.

5.4.2. Experimental Results

Our experiments can be grouped in two categories: i) detecting changes induced in an XML document, w.r.t. a reference grammar, and ii) detecting changes produced in an XML grammar, w.r.t. a valid reference document. Due to space limitations, we selected meaningful changes, which are described as follows.

5.4.2.1. Detecting Changes in XML Documents

Similarly to our classification experiments, we utilized XML data from the online XML version of the ACM SIGMOD Record. Among the various experiments conducted, we present the results obtained when modifying documents *Ord_234.xml*, *Pro_172_2.xml* and *Sigmod_11.xml* [1], evaluating their similarities w.r.t. each of their corresponding grammars respectively. Graphs in Fig. 18.b show the average similarity values obtained when testing each type of modification operation, considering the different modifications described in the previous section. Results show that:

i. Similarity varies linearly w.r.t. the modifications induced via the update operation, going from 1 (0% updates) to 0 (100% updates, i.e., when all document nodes have been relabeled),
ii. Similarity varies linearly w.r.t. the modifications induced via the insertion operation, going from *1* (0% insertions) to ≈ 0.5 (100% insertions, indicating that a structure, which size is equal to that of the original document, has been inserted under the document root node).
iii. The case of the deletion modification operation is special, in that similarity values are not perfectly linear w.r.t. the amount of modifications, and sometimes even increase with the increase of the amount of deletions (which might seem counter-intuitive). That is due to the presence of optional and repeatable elements in the reference grammar, which are sometimes better satisfied (i.e., higher similarities are obtained) after deleting certain elements in the documents. This explains the increase in similarity values in the final stages of the modification process, i.e., at thresholds ≥ 80% (grammar roots encompassing optional siblings such that the deletion of their document counterparts, along with their sub-trees, positively affects the similarity evaluation process).

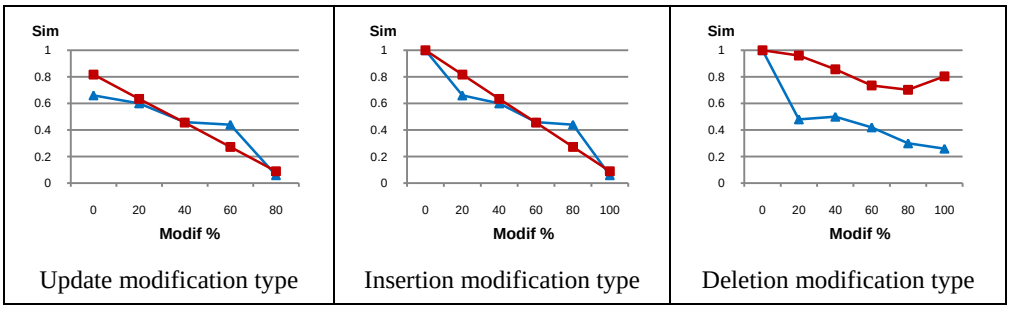
In addition to document modification, we also conducted a set of experiments to detect the changes induced in XML grammars. We particularly modified grammars *OrdinaryIssuePage.dtd*, *ProceedingsPage.dtd* and *SigmodRecord.dtd* (of the online version of the ACM SIGMOD Record), comparing them to documents *Ord_234.xml*,

[1] Recall that *Sigmod_11.xml* results from the decomposition of document *SigmodRecord.xml*.

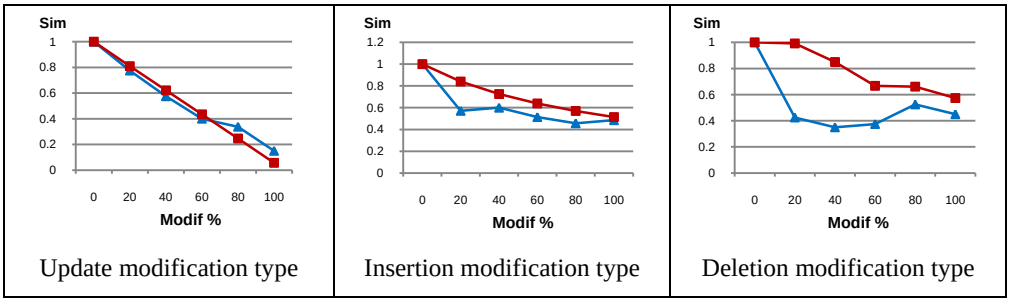
Pro_172_2.xml and *Sigmod_11.xml* respectively. Similarity graphs are similar to those in Fig. 18.b, highlighting a straight correspondence between similarity and modification levels. Graphs are omitted here (and can be found in [73]).

5.4.2.2. User Rankings

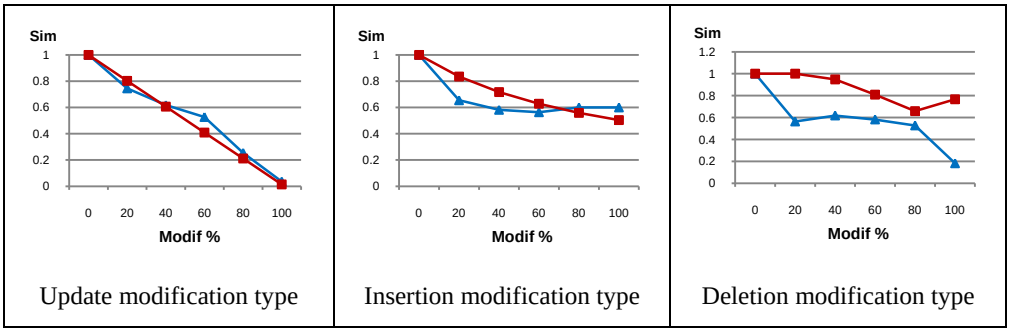
In addition to system generated results, we conducted manual (user) rankings to identify the correspondence between: i) the user's perception of similarity and ii) system-generated similarity scores, in detecting changes in documents/grammars. Similarity graphs corresponding to each of the charts in Fig. 18.b, are shown in Fig. 19.



a. Comparing user and system similarity scores when gradually modifying *Ord_234.xml*.



b. Comparing user and system similarity scores when gradually modifying *Ord_172.xml*.



c. Comparing user and system similarity scores when gradually modifying *Ord_172.xml*.

Fig. 19. Similarity graphs contrasting system and user-generated similarity scores when comparing XML documents *Ord_234.xml*, *Pro_171_2.xml* and *Sigmod_11.xml*, w.r.t. XML grammars *OrdinaryIssuePage.dtd*, *ProceedingsPage.dtd* and *SigmodRecord.dtd*.

Here, user rankings (i.e., *A*, *B*, *C*, …, *F*) were transformed into numerical values so as to be comparable to system generated scores, such as: *A* = 100% (the document conforms to the grammar), *B* = 80% (high similarity), *C* = 60%, *D* = 40%, *E* = 20%, and *F* = 0% similarity (the document and grammar seem completely different to the user). Average correlation scores for each kind of document modification (document node update, insertion, and deletion) in Fig. 19 are show in Table 6.a. Results confirm the relevance of system generated scores w.r.t. the users' perception of similarity: correlation is > 75%, on average, for all kinds of document modifications.

Similarly to the document modification experiments, manual user rakings conducted w.r.t. grammar modification experiments reveal a straight correlation with system generated similarity scores. Similarity graphs are akin to those presented in Fig. 19 and thus are omitted here (they can be found in [73]). Average correlation scores for each kind of grammar modification (grammar node update, insertion, and deletion) are show in Table 6.b. Results confirm the relevance of system generated scores w.r.t. the users' perception of similarity: correlation is > 85%, on average, for all kinds of grammar modifications.

Table 6. Statistical dependency: Pearson Correlation Coef. (*PCC*) between system and user-generated similarity rankings.

a. PCC with document modification experiments.

	Update	Insertion	Deletion
Ord_234.xml	0.9529	0.6962	0.7678
Pro_172_2.xml	0.9891	0.8832	0.4689
Sigmod_11.xml	0.9891	0.8233	0.5869

b. PCC with grammar modification experiments.

	Update	Insertion	Deletion
OrdinaryIssuePage	0.9435	0.9217	0.8575
ProceedingsPage	0.9058	0.8706	0.8992
SigmodRecord	0.8659	0.7679	0.7991

To sum up, experimental results underline our method's effectiveness in accurately discerning modified documents and/or grammars w.r.t. their original versions. Results show a close correspondence between i) the amount of modifications in documents/grammars, ii) system-generated similarity levels, and iii) user-generated similarity rankings, and thus reflect our approach's efficiency in accurately comparing and ranking documents/grammars based on their resemblances/differences, in accordance with the user's perception of similarity.

5.5. Evaluating Intelligent Behavior: Comparing Non-Conforming Documents and Grammars

In addition, we evaluate our method's quality in intelligent behavior (noise resistance), i.e., its ability to identify (disregard) documents which are similar (different) to the reference grammars, such that none of the documents actually conforms to any of the grammars. This corresponds to the most practical case on the Web, where the system user/administrator does not have prior knowledge about the XML documents scattered online, and would like to identify those which approximately validate (most likely correspond to) her predefined grammars.

5.5.1. Experimental Scenario

To simulate the process of comparing non-conforming documents and grammars, we combine both grammar transformation (modification) and document classification methods. Considering an XML document collection with a set of documents (e.g., D_1 and D_2) conforming to predefined reference grammars (e.g., G_1 and G_2), we first deliberately introduce certain amounts of modifications in the grammars (inserting, deleting and/or relabeling certain amounts of grammar nodes, as described in the experimental scenario of Section 5.4). Subsequently, we compare the XML documents with the modified grammars (G_1' and G_2'), performing XML classification (as described in the experimental scenario of Section 5.3). In other words, given a set of modified grammars, we attempt to identify those documents which are similar to the grammars, given that the documents are not written exactly in those grammars (e.g., our objective is to effectively classify D_1 under G_1', and D_2 under G_2', since they are probably similar, despite the fact that neither documents were written for those grammars, but rather conform to their original versions: G_1 and G_2).

Classification metrics are adapted from section 5.3.1 as follows. Given an original grammar G_i, its modified version G_i', and an extracted class K_i corresponding to the modified grammar G_i':

- a_i is the number of XML documents in K_i that indeed conform to the original grammar G_i. These are the documents which are most probably similar to the current modified version G_i'.
- b_i is the number of documents in K_i that did not originally conform/correspond to G_i. In other words, these documents are less likely to be similar to G_i'.
- c_i is the number of XML documents not in K_i, although they conform to the original grammar G_i. In other words, these documents are probably similar to the modified grammar G_i' and should have been classified in K_i.

Consequently, *precision*, *recall* and *f-value* are computed following a_i, b_i, and c_i, using formula (6) in section 5.3.1.

5.5.2. Experimental Results

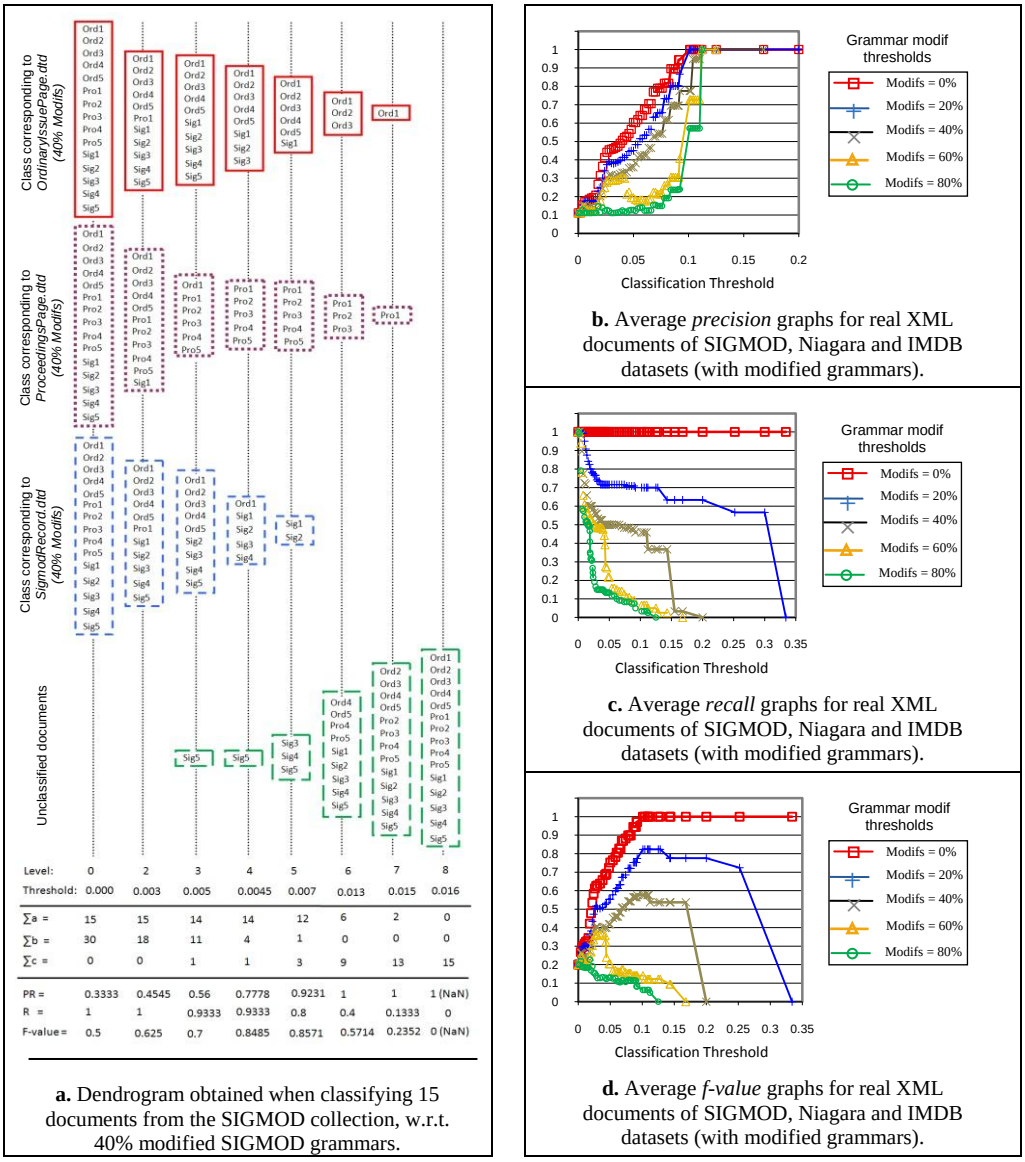
We consider the real XML data sets described in Table 3 (i.e., SIGMOD, Niagara, and IMBD). Each of the corresponding grammars is modified with increasing transformation thresholds, ranging from 0% (original grammars), to 20%, 40%, 60% and 80% w.r.t. the original grammars. We combine all three transformation operations: *insertion*, *deletion*, and *relabeling* (cf. Section 5.4) in inducing modifications. Our experiments are based on multilevel classification, similarly to the experiments in Section 5.3.

The central difference here is that since none of the documents actually conforms to the class reference grammars, all XML documents will be eventually filtered out of the predefined classes (whereas in the experiments of Section 5.3, the classes always contain the XML documents conforming to their reference grammars). A sample dendrogram structure depicting the classification of non-conforming XML documents and grammars is depicted in Fig. 20.a. Average *precision*, *recall* and *f-value* results are depicted in Fig. 20.b, Fig. 20.c and Fig. 20.d respectively.

On one hand, when multilevel classification is applied on conforming documents/grammars (cf. Section 5.3), *recall* is constantly equal to 1, indicating that all documents are successfully identified in the 'correct' classes (i.e., classes corresponding to their conforming grammars, having $Sim_{XDoc\_XGram}=1$). However, when classifying non-conforming documents/grammars (Fig. 20), *recall* varies from 1 (maximum value, where all documents are identified in the correct classes, corresponding to the initial classification step, Fig. 20.a) to 0 (minimum value, where all documents are misclassified, attained in the final classification step). That is due to the fact that Sim_{XDoc_XGram} is always $\neq 1$ in this case, since none of the documents conforms to the grammars.

On the other hand, results in Fig. 20.a, b and c show that *precision*, *recall* and *f-value* respectively decrease while increasing the grammar modification threshold. In other words, classification accuracy steadily decreases when the resemblance/relatedness levels between the documents and the grammars decrease (simulated, in our experiments, by varying/increasing the grammar modification threshold). Note that we do not show the results of *Modif*=100% in Fig.

20, since it underlines the case where all grammars are completely different from all documents, which contradicts the idea of using grammars as document classification references to compute *precision* and *recall*.



a. Dendrogram obtained when classifying 15 documents from the SIGMOD collection, w.r.t. 40% modified SIGMOD grammars.

b. Average *precision* graphs for real XML documents of SIGMOD, Niagara and IMDB datasets (with modified grammars).

c. Average *recall* graphs for real XML documents of SIGMOD, Niagara and IMDB datasets (with modified grammars).

d. Average *f-value* graphs for real XML documents of SIGMOD, Niagara and IMDB datasets (with modified grammars).

Fig. 20. Classification results obtained with non-conforming documents/grammars.

To sum up, while exact (Boolean) XML document validation methods (cf. State of the Art in Section 6.2) could be used to perform document classification in the case of conforming XML documents/grammars (identifying which documents conform to which grammars), such methods become obsolete (i.e., completely ineffective) when non-conforming documents/grammars come to play. In such a context, an 'intelligent' approximate similarity evaluation method (such as the one proposed in this study) becomes crucial.

Note that while our current experimental study conveniently exploits XML grammars as references to evaluate result quality (computing *precision* and *recall* accordingly), we are currently building a larger XML benchmark better suited to the experimental tasks, aimed at making use of blind testing. Here, we were unable to utilize the current INEX[1] data set in our experiments since it targets XML textual similarity (i.e., similarity between element/attribute values made of long text fields) which is out of the scope of this study (here, as mentioned earlier, we focus on XML structure, i.e., document/grammar element/attribute tag names and their structural positions, and disregard values).

5.6. Comparative Study

An experimental study, comparing the effectiveness of our XML document/grammar comparison method with existing approaches, would have been interesting, and would have allowed us to further validate our method. Nonetheless, most related studies in the literature (cf. Background in Section 6) do not precisely tackle the issue of ranked document/grammar comparison, but rather handle (Boolean) document validation, or transformation/correction. Other

[1] http://inex.is.informatik.uni-duisburg.de/

methods, e.g., [32, 75], perform some sort of ranked similarity evaluation, yet are based on a specific premise: that the possible distortions between the documents/grammars are known in advance, which makes it difficult to define a common experimental scenario. Thus, we currently settle for a qualitative comparison, depicting the main characteristics, commonalities and differences between our approach and related studies. We also compare our method with its most pertinent predecessor: *DTDMatch* in [9, 10], which to our knowledge is the main existing work to actually allow (knowledge-free) ranked document/grammar similarity evaluation[1].

5.6.1. Qualitative Analysis

Table 7 summarizes the main differences between our method and related approaches. In short, our approach is: i) fine-grained, extending the concept of tree edit distance distance (as an efficient technique for comparing XML-based structured data [17]) to detect and identify the structural similarities and disparities between XML documents and grammars, taking into account the most common XML grammar constraints, namely *MaxOccurs* and *MinOccurs*, repeatable expressions and recursive declarations, which are partly disregarded in the main existing methods, e.g., [7, 10, 32, 57], ii) producing a ranked similarity result, i.e., a similarity value $\in [0, 1]$ interval, in comparison with existing Boolean (validation) methods, e.g., [7, 8, 49], which only provide a *Boolean* output, iii) capable of identifying (in the course of computing the similarity value) an edit script transforming the XML document into one conforming to the XML grammar, similarly to XML transformation methods [57] (note that in our current paper, we do not provide the additional algorithms needed for extracting edit scripts. We report this issue to an upcoming dedicated study).

Table 7. Comparing our method to related approaches.

Approach	Performs Document Validation	Generates Edit Script	Computes similarity value	Considers element constraints	Considers repeatable expressions	Considers recursive declarations	Dedicated to XML Grammars	Complexity level
Segoufin et Vianu [49]	✓	×	×	✓	✓	✓	✓ (DTD)	$O(Exp(\|G\|))$
Barbosa *et al.* [8] [2]	✓	×	×	✓	✓	×	✓ (DTD/XSD)	$O(\|D\|\times log(\|G\|))$
Balmin *et al.* [7] [3]	✓	×	×	✓	×	×	✓ (DTD/XSD)	$O(\|D\|\times log(\|G\|))$
Bouchou *et al* [15]	✓	*Partial*	*Partial*	✓	*'And' only* [3]	×	✓ (DTD/XSD)	$Exp(MaxDeg(\|S\|))$
Suzuki [57]	×	✓	×	✓	×	×	✓ (DTD)	$O(Exp(\|G\|))$
Bouchou *et al.* [14]	✓	×	×	✓	✓	×	✓ (DTD/XSD)	$O(\|D\|\times log(\|G\|))$
Xin G. [79] [3]	✓	✓	✓	×	×	✓	✓ (DTD)	$O(\|D\|\times\|G\|\times\log(\|G\|))$
Grahne & Thomo [32]	✓	×	✓	×	×	×	× (struct data)	$O(\|A_G\|\times N\times\|P\|^3)$
Thomo *et al.* [50, 75]	✓	×	✓	×	*Partial*	×	✓ (DTD/XSD)	$O(K^3\times N^{2(K+1)}\times\|P\|)$
Bertino et al. [9, 10] [2]	✓	×	✓	✓	✓	×	✓ (DTD)	$O(\Gamma^3\times(\|D\|+\|G\|))$
Our Approach	✓	✓	✓	✓	✓	✓	✓ (DTD/XSD)	$O(\|N_G\|\times\|D\|\times\|G\|)$

$|D|$: cardinality of the XML document
$|G|$: cardinality of XML grammar
$|A_G|$: number of states in the automaton describing the data-grammar,
N: cardinality of semi-structured document,
$|P|$: number of states in the distortion transducer,
K: number or allowable transformation operations
Γ: maximum node fan-out in document tree D
N_G: number of conjunctive grammars representing G
(cf. State of the Art Section 6 for details)

5.6.2. Comparison with DTDMatch

In the following, we compare our approach with its most pertinent predecessor: *DTDMatch* [9, 10].

1. Both *DTDMatch* and our approach model XML documents and XML grammars as labeled trees. *DTDMatch* follows an *intentional* approach in producing one compact tree representation per grammar *G*, representing the set of rules constraining the content of each element in the grammar (i.e., describing the language *L*(*G*) of the grammar), while we follow a s*emi-intentional* approach such that each grammar *G* is represented as the set of conjunctive trees $\{C\}_G$, where the grammar's expressiveness (language) is distributed among its constituent conjunctive grammars, $L(G) = \bigcup_{C_i \in \{C\}_G} L(C_i)$ (cf. Fig. 21.a).
2. While *DTDMatch*'s grammar tree representation is more compact, nonetheless our grammar tree representation is more similar (in its structural properties) to XML document tree representation, as it was designed for this specific purpose in order to simplify the document/grammar tree comparison process.
3. *DTDMatch* only considers (context-free) DTD grammars whereas our approach processes (context-sensitive) XSD grammars, with more expressive power related to *MinOccurs* and *MaxOccurs* constraints.
4. *DTDMatch* does not discuss the case of recursive declarations, whereas *strong linear recursive declarations* are handled in our grammar tree representation model.

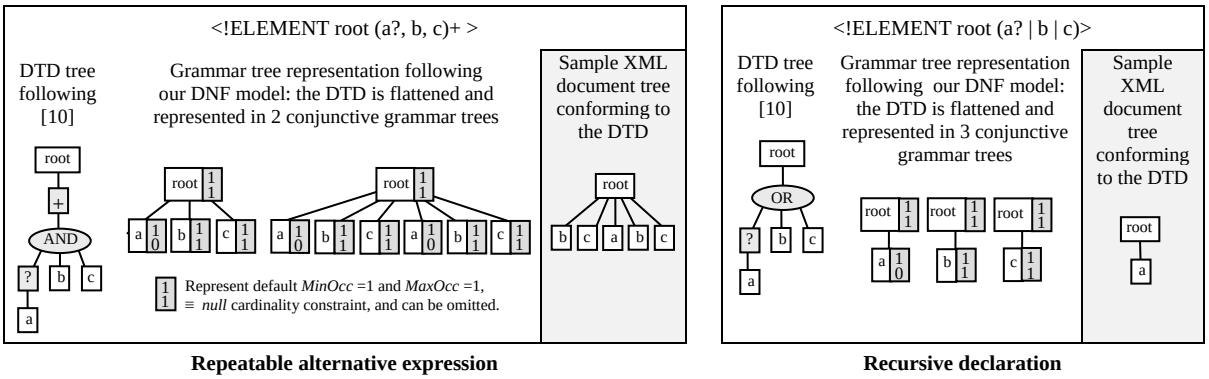
---

[1] We do not empirically compare our method with *DTDMatch* since the authors do not provide the detailed algorithm/code of the method.

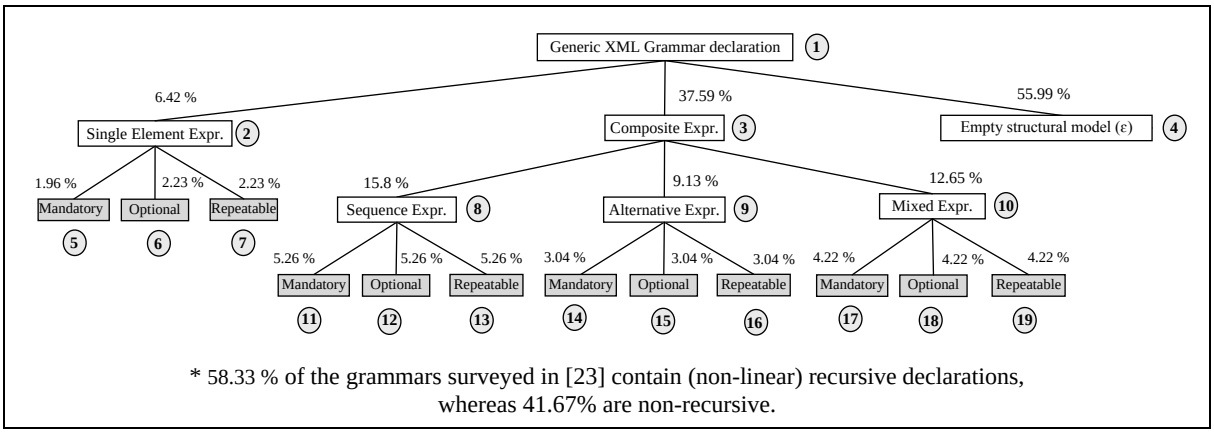
[2] Note that some studies, such as [7, 8, 79], support arbitrary regular expressions as production rules in the grammars. Nonetheless, [7, 79] do not discuss repeatable expressions, while [7, 8] do not mention recursive declarations.

[3] Considers repeatable sequence expressions only (i.e., expressions connected via the *And* operator) and disregards repeatable alternative expressions (connected via the *Or* operator).

5. The *DTDMatch* algorithm consists of mapping functions which allow to identify: i) elements appearing both in the document and in the DTD (*common elements*), ii) elements appearing in the document but not in the DTD (*plus elements*), iii) and elements appearing in the DTD but not in the document (*minus elements*), assigning different weights to each group of elements to tune the similarity measure following the user's needs. Our approach assigns no such weights. Nonetheless, by post-processing the edit script describing the transformations applied on a document tree so that it becomes valid w.r.t. the grammar tree, such (*common*, *plus*, and *minus*) element mapping can be identified (e.g., such an approach is developed in [70], in the context of XML grammar tree matching).



a. Disparities in tree representations between XML document and grammar structures, following the grammar (DTD) tree representation model in [10] (one of the central methods in the literature).



b. XML grammar expression usage probability tree (averages and/or extrapolated from [11, 23, 38]).

Fig. 21. Comparing our approach with *DTDMatch* [10].

6. The *DTDMatch* algorithm can produce wrong matches when a certain assumption holds: *in the declaration of an element, two sub-elements with the same tag are forbidden* (e.g., declaration *<!ELEMENT root(b, b, c)>* is not allowed since *b* appears twice). Following our algorithm, wrong matches can also be obtained when grammar transformation *Rule 1* is applied (in simplifying grammar expressions), $(A_x^y)_u^v \rightarrow^{R1} A_{x\times u}^{y\times v}$ since it does not verify the *ISP* (*Information Structure Preserving*) property, and thus might produce transformed grammar expressions that do not preserve the same expressiveness (language) of their original counterparts.

7. The *DTDMatch* algorithm considers tag similarity, i.e., handling the possibility that tags might be syntactically different by with semantically similar meanings. This issue is not discussed in our current study, and is reported to a future extension of this work. This can be performed through the investigation of alternative tree operations cost models (varying costs w.r.t. the semantic relatedness between document and grammar node labels given a semantic reference such as Wordnet [45], Wikipedia [84], or Google [37]), similarly to the studies in [63, 66, 70].

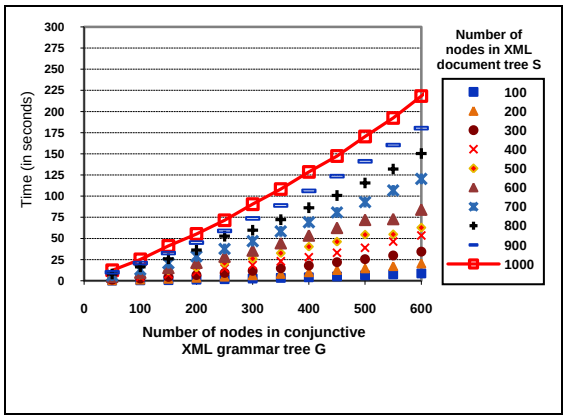
In short, our approach builds on *DTDMatch* in different aspects, attempting to handle more expressive XSD constraints, and produced an improved method. Note that we are currently conducting a case study on a large set of real DTD and XSD grammars (mainly acquired from survey in [11, 23, 38]) in order to estimate empirical usage probabilities concerning the different kinds of XML grammar declarations (namely *MinOccurs* and *MaxOccurs*, repeatable expressions, multiple identical sub-elements, and recursive declarations) found in real-world grammars. This would help us better evaluate the performance levels of our approach in comparison with its most relevant predecessor(s) in terms of: the percentages of correct grammar expression matches, wrong matches, exact validation

ratio, approximate validation ratio, etc., considering the usage probabilities of grammar expressions involved (Fig. 21.b provides a glimpse on our preliminary usage probability estimates).

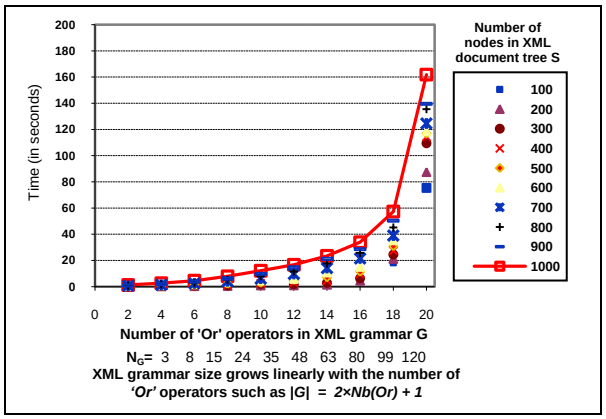
5.7. Timing and Space Analysis

Timing experiments were carried out on a *Dell Precision* system with an *Intel* 2.53 GHz processor and 4 GB RAM. In Section 4.5, we have shown that the complexity of our approach is of $O(|S|+|G|+(N_G \times |S| \times |C_G|))$ time, and simplifies to typical quadratic $O(|S| \times |G|)$ time, w.r.t. the sizes of the XML document and XML grammar being compared, and worst case $O(N_G \times |S| \times |G|)$ considering the number of conjunctive grammars involved.

We start by verifying our approach's quadratic dependency on the combined XML document and grammar sizes, i.e., $O(|S| \times |G|)$, which equally underlines a linear dependency on each of the document and grammar sizes. Fig. 22.a (left graph) shows that the time to identify the structural similarity between XML document trees and conjunctive XML grammar trees of various sizes grows in an almost linear fashion with tree size.

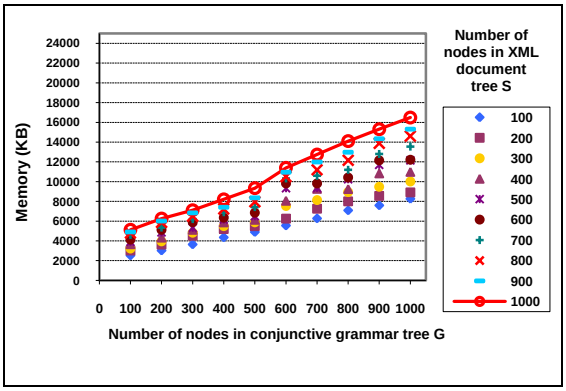


Time to compute similarity between XML document and conjunctive grammar trees

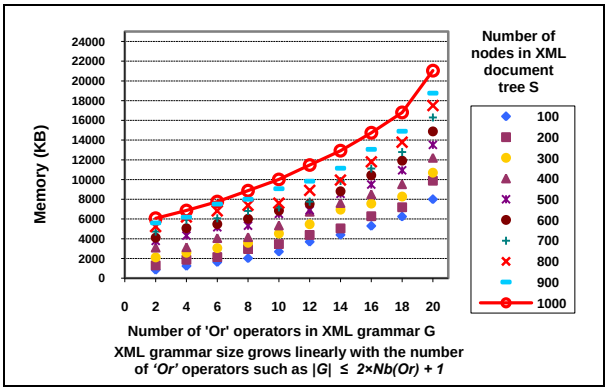


Varying the number of *'Or'* operators (number of conjunctive grammars) **in the XML grammars**

a. Timing results for our XML document/grammar comparison method.



Memory required to compute similarity between XML document and conjunctive grammar trees.



Varying the number of *'Or'* operators (number of conjunctive grammars) **in the XML grammars**

b. Memory usage for our XML document/grammar comparison method.

Fig. 22. Time and space results.

Second, we varied the number of *Or* operators in the grammar expressions, in order to deliberately vary the number of potential conjunctive grammars (N_G) and study its effect on overall complexity. We considered different configurations, using: alternative expressions (of the form *(A | B | C |...)*) and mixed expressions (of the form *(A | B)* , *(C | D)*, ...). Fig. 22.a (right graph) shows the worst case time results (mainly obtained when using mixed expressions). The time curve remains mostly linear w.r.t. tree size even when the *'Or'* operator comes to play, and only starts to grow faster with XML grammar size when the number of concatenated *Or* expressions (e.g., expressions of the form (*A* | *B*), (*C* | *D*)…) surpasses 16 (i.e., yielding more than 80 conjunctive grammar trees per single XML grammar). However, note that such a huge number of mixed *Or* expressions is unlikely to appear in real XML grammars [11, 23, 38] (around 12.65% of grammar expressions combine *And* and *Or* operators, cf. Fig. 21.b – node 10).

A mathematical analysis regarding the variation of the number of conjunctive grammars (N_G) w.r.t. different configurations of *Or* operators is provided in Appendix I. Results concur with Fig. 22.a (right graph), showing that the most common *alternative* expressions generate a number of conjunctive grammars linear in the number of *Or* operators, while certain specific cases (of usually *mixed*: *And*-*Or* expressions) yield polynomial and/or exponential N_G.

Similarly to time complexity, memory usage results in Fig. 22.b show that our approach is polynomial in the combined size of the XML document and grammar trees being compared. It is almost linear in the size of each of the XML document and conjunctive grammar trees (Fig. 22.b – left graph), and becomes slightly quadratic w.r.t. grammar size when *'Or'* operators come to play (ranging from 2 to 20 *alternative* - '*Or*'- expressions per grammar, cf. Fig. 22.b – right graph). Recall that conjunctive grammar trees consist of references (pointers) to the elements/attributes in the source grammar, and thus require limited space in comparison with the actual document and grammar sizes (cf. complexity analysis in Section 4.5.2). This underlines the limited increase in space (in comparison with a greater increase in time, as shown in Fig. 22.a), even with a large increase of up to 20 *alternative* expressions per grammar.

6. State of the Art

In the following, we briefly review the literature on XML document/grammar comparison (a detailed review has been published in [62]). We also briefly discuss our own research activities related to the problem, in order to better highlight the contributions of this paper. The interested reader can also refer to [4, 17, 69] for reviews and comparative studies concerning XML document comparison, and [29, 51] for comprehensive reviews on the state of art in XML grammar matching, which are also related to the hybrid task of document/grammar comparison.

Comparing XML documents with XML grammars has been explored in several domains. Various methods have been proposed for validating XML documents against XML grammars [7, 8, 14, 22, 36, 49]. Methods dedicated to XML document-to-grammar transformation and correction have been investigated in [15, 20, 57]. Yet, to our knowledge, the main methods (different from ours) to address the issue of document/grammar similarity evaluation, i.e., producing a similarity score (allowing ranked similarity results), are provided in [32, 75] and [9, 10].

6.1. Approximate Pattern Matching with VLDC

An intuitive XML document/grammar comparison solution could be explored in terms of approximate matching with the presence of *Variable Length Don't Cares* (*VLDC*). A *VLDC* symbol (e.g., $\wedge$) in a string pattern may substitute for zero or more symbols in the string [3, 39]. Approximate *VLDC* string matching means that, after the best possible substitutions have been made, the pattern still does not match the data string and thus a matching distance is computed. For example, "*comp* $\wedge$ *ng*" matches "*commuting*" with distance 1 (i.e., the cost of removing the "*p*" form "*comp* $\wedge$ *ng*" and having the "$\wedge$" substitute for "*mmuti*"). The *VLDC* problem has been generalized for trees [81], introducing *VLDC* substitutions for paths or sub-trees. Yet, *VLDC* symbols are different from operators XML grammars operators: *VLDC* symbols can replace any string (w.r.t. string matching) or sub-tree (w.r.t. tree matching) whereas the XML grammar operators specify constraints on the occurrence of a particular node (and consequently the sub-tree rooted at that node). For instance, the DTD operator "*?*" associated with a given element *dummy?* designates that the node entitled *dummy* (and not any other) can appear *0* or *1* time. The same applies for all XML grammar operators.

6.2. XML Document/Grammar Validation

XML document validation w.r.t. XML grammars has recently gained attention, as one of the aspects of XML data management [7, 8, 14, 22, 36, 49]. Here, XML grammars are generally viewed as context-free (DTD-like) regular tree grammars [58]. Thus, verifying if an XML document D conforms to XML grammar G comes down to checking whether the document tree is included in the language defined by the grammar, i.e., if $D \in L(G)$. The standard procedure for testing membership in a formal language is to simulate the automaton that accepts the language on the input strings [34]. Hence, XML validation methods, e.g., [7, 8, 14, 49], have investigated different variations to extend automaton-based techniques to deal with the special case of XML regular tree grammars and XML document trees. In general, the validation is performed in $O(|D| \times \log(|G|))$ time, where $|D|$ and $|G|$ designate respectively the sizes of the XML document and XML grammar. The authors in [49] show that the construction of a standard automaton for (streaming) XML validation requires exponential time in the size of the XML grammar, when the latter encompasses recursive declarations. Note that methods for XML document validation generate a *Boolean* result indicating whether the XML document is valid or not w.r.t. the grammar. They do not produce a (ranked) similarity score.

6.3. XML Document Transformation and Correction

Methods for identifying the edit script transforming a given XML document, to another document conforming to a given DTD grammar, have been proposed in [57, 58, 79, 80]. The approach in [57] builds a special graph structure G, based on the XML document tree D, underlining all possible transformation operations applicable to D (i.e., node insertion, deletion and update). The algorithm goes through graph G, and verifies which paths have sequences of labels that satisfy the DTD regular expressions. This is achieved via dedicated NFAs (Non-deterministic Finite Automatons). The proposed method addresses simple DTDs, and does not consider XSD *MinOccurs* and *MaxOccurs* operators, nor does it discuss the special cases of repeatable and recursive expressions. The authors show that their approach is polynomial on document and grammar expression sizes when the cost of an operation on a node only depends on the node label itself, and that it becomes non-polynomial (exponential) otherwise, highlighting a strongly *NP-Complete* decision problem. No experimental evaluation is provided. Another approach to XML document/grammar transformation is provided in [79, 80]. It introduces a tree edit distance method to identify the set of operations transforming the XML document to one conforming to the grammar. However, the author simplifies DTD definitions into data-guide like structures (simulated via hedge automatons [46]), omitting all cardinality and alternativeness

constraint operators. The proposed method is of $O(|D|\times|G|\times \log(|G|))$ time, where $|G|$ is the size of the grammar and $|D|$ is the size of the XML document tree.

A problem comparable to that of document-to-grammar transformation is that of document-to-grammar correction [6, 15, 58]. The scenario considered here is that of dynamic XML documents which are modified and updated frequently, underlining the need to continuously test their conformance w.r.t. the corresponding grammars. The authors in [6, 15] propose to correct those sub-trees, in the modified XML document, where validation fails w.r.t. a given DTD grammar. The methods exploit automatons and tree edit distance to identify the set of possible sub-tree corrections, such that their distances from the original sub-tree are within a given threshold. The approach is shown exponential in the size of document node fan-out (maximum node degree), and has been exploited to incrementally validate XML integrity constraints defined as XML functional dependencies [13]. In [54, 58] the authors extend document-to-grammar (DTD) correction to deal with more expressive XML schemas, represented as *simple type tree grammars* (where the left-hand side of a production rule may be surrounded by context information, consisting of terminal symbols), representing repairs as sequences of edit operations to alter XML trees. The authors in [52, 53] investigate user-defined XML document adaptations, i.e., sequences of document transformation operations intended to adapt documents valid for an original grammar G to a new grammar G'. The objective is to check whether a user-proposed document *adaptation* is guaranteed to produce a document valid for the new grammar, in order to avoid the usually expensive revalidation of documents upon grammar modification. Transformation operations are expressed as sequences of XQuery update primitives, automatically inferred from the original grammar using a Hedge automaton and a set of rules describing each operation type (e.g. rename node, insert as first child node, insert as last child node, etc.). Type inclusion is then used as a conformance test w.r.t. the types of updates extracted from the updated schema G'.

While the methods in [6, 15, 57, 58, 79] produce transformation and correction scripts, they do not however address the issue of XML document/grammar similarity.

6.4. XML Document/Grammar Similarity

Very few approaches have been developed to measure the structural similarity between XML documents and grammars. The main methods are provided by Thomo *et al.* [32, 50, 75] and Bertino *et al.* [9, 10].

In [32], the authors address the problem of determining whether semi-structured data conform to a given data-guide, in the context of approximate querying. The authors define a distortion transducer through which the data-guide can be distorted via elementary transformations (e.g., node insertions, deletions and updates) and then test if the database conforms to the resulting data-guide. The same technique is exploited to compare semi-structured data with a given query. The approach in [32] is developed for generic semi-structured data and data-guides, rather than for XML documents, and does not consider any of the XML grammar repeatability and alternativeness constraints. In a more recent study [50, 75], the authors propose a similar approach toward approximate XML validation. Dedicated pushdown transducers are designed to modify XML grammars by a predefined tolerable number of transformation operations (e.g., node insertions, deletions and updates) and then test if the XML documents conform to the resulting grammar. Both methods require typical polynomial time with simple (data-guide like) grammar structures, e.g., $O(K\times N^2\times|P|)$ where K is the number of allowable transformation operations, N is the size of the alphabet (XML document), and $|P|$ the number of states in the distortion transducer P [50, 75]. However, the authors show that complexity becomes exponential when considering intrinsic XML grammar properties, namely repeatable elements. They do not discuss optional, alternative or recursive declarations.

Note that methods in [32, 50, 75] are based on the assumption that the possible derivations from the original grammar specification are pre-designed by the user through the transducer (pre-processing phase). While this might be feasible in specific applications such as the fast validation of streaming (homogeneous) XML, yet, it is a limitation to the general problem of XML document/grammar comparison (namely approximate validation of heterogeneous XML data) where no prior knowledge of the possible deviations is known in advance.

To our knowledge, the main approach that specifically addresses the general problem of XML document/grammar similarity, particularly DTDs, was proposed by Bertino *el al.* in [9, 10]. Here, XML documents and DTDs are modeled as labeled trees, with additional nodes for representing cardinality and alternativeness operators (i.e., *?*, *, +, *And*, *Or*). The proposed algorithm (originally proposed in [10], and formalized in [9]) exploits dedicated measures to consider the level (i.e. depth) in which the elements occur in the hierarchical structure of the XML and DTD tree representations, as well as element complexity (i.e. the cardinality of the sub-tree rooted at the element) when computing similarity values. The algorithm relies on the identification and evaluation of: i) elements appearing both in the document and in the DTD (common elements), ii) elements appearing in the document but not in the DTD (plus elements), iii) and elements appearing in the DTD but not in the document (minus elements). Different weights can be assigned to each group of elements to tune the similarity measure following the user's needs. The proposed algorithm is of typical polynomial time complexity ($O(\Gamma^2\times(|D|+|G|)$) where $|D|$ and $|G|$ underline XML document tree and DTD tree cardinalities respectively, and Γ the maximum node fan-out in the XML document tree), especially when a certain assumption holds: *in the declaration of an element, two sub-elements with the same tag are forbidden* (e.g., declaration *<!ELEMENT root(b, b, c)>* is not allowed since *b* appears twice). The authors discuss that when the above assumption does not hold, the algorithm can become of worst case exponential complexity, and can produce *wrong matches* between document and DTD elements (i.e., the minimality of the similarity is no longer guaranteed).

6.5. Our Own Research Activities related to XML Similarity

Part of our research activities have been focused around the study of XML similarity, developing measures for comparing i) XML documents, ii) XML grammars, and iii) XML documents and grammars, and their application in

specific real-world scenarios. We have largely focused on XML structure, and have recently tackled XML content (specifically in the context of RSS merging and SOAP multicasting). Various results have been accomplished, mainly:

- Extending the tree edit distance algorithm in [48] so as to detect structural similarities and repetitions amongst XML sub-trees [65], and XML leaf nodes [67], previously unaddressed in existing approaches,
- Integrating semantics (i.e., considering the meanings of XML labels, via semantic networks such as WordNet [45]) in the structural comparison of XML documents [64]. An improved method, considering XML sub-tree structural and semantic similarities, has been recently published in [66].
- Developing a fine-grained method for XML grammar comparison [63], considering element semantic and syntactic similarities, cardinality and alternativeness constraints, as well as data-types and ordering. An extended study with detailed theoretical and experimental analyses has been recently proposed in [70].
- Quantifying the similarity and identifying the relations (i.e., inclusion, intersection, disjointness, and equality) among XML element content values, particularly among RSS items [60], developed toward RSS merging [59].
- Developing a filter-differencing framework for SOAP multicasting, identifying the common pattern and differences between SOAP messages, modeled as XML trees, to multicast similar messages together [71, 72].

Our only proposal aimed at dealing with XML document/grammar comparison was presented in [68], in addition to a review paper published in [62]. Similarly to our current study, the approach in [68] is based on the concept of tree edit distance as a more effective solution to comparing XML trees. The approach targets DTD grammars, and considers basic DTD cardinality and alternativeness constraints (i.e., ?, +, *, *And*, *Or*). Hence, it must be viewed as the groundwork for the general approach in this paper. Here, we aim to consider more expressive XSD operators (e.g., *MinOccurs* and *MaxOccurs*), including repeatable expressions and recursive declarations (omitted in [68]).

6.6. Discussion

To sum up, various methods have been proposed for XML document/grammar validation, e.g., [7, 8, 36] and transformation/correction, e.g., [6, 15, 58]. Yet, most approaches do not address the issue of document/grammar similarity evaluation and do not produce a similarity score. Those few methods developed for XML document/grammar similarity are either generic (disregarding most grammar constraints) and intended to consider pre-designed derivations [32, 75], or developed for the DTD (context-free) grammar language and do not consider XSD (context-sensitive) structure and constraints which are more complex and expressive (e.g., *MinOccurs* and *MaxOccurs*) [9, 10].

Some methods in the context of XML grammar matching [40, 56] have proposed to reduce (sacrifice) XML grammar expressiveness to simplify the comparison task, using simplification rules to eliminate repeatable and alternative expressions (e.g., transforming the *Or* operator into an *And* operator: $(A \mid B) \rightarrow (A, B)$, brute-force flattening of repeatable expressions such as: $(A, B)+ \rightarrow (A+, B+)$, $(A, B)* \rightarrow (A*, B*)$, etc.). While such rules seem practical in simplifying XML grammars, yet, they reduce grammar expressiveness, which in turn yields erroneous similarity results, and thus not in line with our goal of providing a fine-grained method to XML document/grammar comparison.

7. Conclusion

In this paper, we propose a structure-based similarity approach for comparing XML documents and XML grammars (DTDs and/or XSDs), performing approximate structure XML validation. The proposed approach has several applications, including document classification, transformation, and XML selective dissemination (e.g., user profiles being represented as grammars against which the documents will be matched). Based on the tree edit distance concept, our approach takes into account the most common XML grammar operators that designate constraints on the existence, repeatability and alternativeness of XML elements/attributes, namely *MinOccurs*, *MaxOccurs*. It produces similarity values in *[0, 1]* interval (in comparison to *Boolean* output obtained with classic XML validation methods). Also, an edit script can be generated from the edit distance computations which can describe the changes required to transform an XML document into one conforming to the grammar (which is central for document transformation and correction applications). Note that our XML grammar tree model considers complex declarations, including: i) repeatable sequence expressions, ii) repeatable alternative expressions, and iii) recursive expressions, which have been previously disregarded in most existing approaches, e.g., [9, 32, 57]. In addition, it is not limited to context-free (DTD-like) grammar declarations: where the definition of an element is unique and independent of its position in the grammar; but can be used with context-sensitive (XSD-based) declarations: where identically labeled elements can have multiple definitions in different contexts in the grammar. Our theoretical and experimental results showed that our approach yields accurate structural document/grammar similarity results (characterized by high structural document classification and ranking quality).

We are currently extending our approach to consider, not only the structural properties of XML documents and grammars, but also the semantic similarities between XML element/attribute node labels (given a reference semantic information source such as WordNet [45], Wikipedia [84], or Google [37]), through the investigation of alternative tree edit operations cost models similarly to the studies in [63, 66, 70]. In the near future, we plan to extend our method to consider XML element/attribute tag names as well information contents (element/attribute values). By adding additional constraints on the data content of elements/attributes, grammars could be exploited as *content-and-structure* queries, taking into account the structure of XML data in the search process, and returning ranked answers as in information retrieval (IR). This would also give rise to more elaborate content models, such elements defining hyper-links (IDREFS or XLink), which would require dedicated graph-based comparison functions. Another direction is the

extension of our grammar tree model to handle unordered XML document trees, i.e., XML trees where only ancestor relations are considered to be significant in the XML structure, which might be more suitable for various database applications such as document clustering and pattern discovery [24, 41]. Note that combining database (DB) structural "binary answer" XML search (e.g., XML-QL and XQuery) and information retrieval query result ranking (e.g., approximate XML validation), is a prominent trend in both DB and IR research.

Acknowledgements

This work is supported in part by: the Research Support Foundation of the Sate of Sao Paulo (FAPESP Post-doc Fellowship n# 2010/00330-2), STIC AmSud project Geo-Climate XMine, the CEDRE research collaboration program, project *AO 2011 "Easy Search and Partitioning of Visual Multimedia Data Repositories"* (funded by the French National Center for Scientific Research – CNRS, and the Lebanese CNRS), and by the ACM French Chapter on Applied Computing SIGAPP.*fr*.

References

[1] Abiteboul S. *et al.*, *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann Publisher (1st Edition), 1999. pp. 258.
[2] Abu-Ghazaleh N. *et al.*, *Differential Serialization for Optimized SOAP Performance.* Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC'04), 2004. pp. 55-64.
[3] Akatsu T., *Approximate String Matching with Don't Care Characters.* Information Processing Letters, 1995. (55):235-239.
[4] Algergawy A. *et al.*, *XML Data Clustering: An Overview.* ACM Computing Survey 2011. 43(4):25.
[5] Algergawy A. *et al.*, *XML schema element similarity measures: A schema matching context.* Proceedings of the 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2009), 2009. pp. 1246-1253, Portugal.
[6] Amavi J. *et al.*, *On Correcting XML Documents with Respect to a Schema.* The Computer Journal, 2013.
[7] Balmin A. *et al.*, *Incremental validation of XML documents.* ACM Transactions on Database Systems, 2004. 29(4):710-751.
[8] Barbosa D. *et al.*, *Efficient Incremental Validation of XML Documents.* Proceedings of Inter. ICDE Conf., 2004, pp. 671-682.
[9] Bertino E. *et al.*, *Measuring the structural similarity among XML documents and DTDs.* Journal of Intelligent Information Systems, 2008. 30(1):55-92.
[10] Bertino E., Guerrini G., and M. Mesiti, *A Matching Algorithm for Measuring the Structural Similarity between an XML Documents and a DTD and its Applications.* Elsevier Information Systems, 2004. (29):23-46.
[11] Bex G. J., Neven F., and Bussche J.V., *DTDs versus XML Schema: A Practical Study.* International Workshop of the Web and Databases (WebDB'04), 2004. pp. 79-84.
[12] Bille P., *A Survey on Tree Edit Distance and Related Problems.* Theoretical Computer Science, 2005. 337(1-3):217-239.
[13] Bouchou B., Alves M., and de Lima M., *A Grammarware for the Incremental Validation of Integrity Constraints on XML Documents under Multiple Updates.* T. on Large-Scale Data- and Knowledge-Centered Systems, 2012. 6:167-197.
[14] Bouchou B. *et al.*, *Efficient Constraint Validation for XML Database.* Informatica, 2007. 31(3): 285-309.
[15] Bouchou B. *et al.*, *XML Document Correction : Incremental Approach Activated by Schema Validation.* Proceedings of the International Database Engineering and Applications Symposium (IDEAS), 2006. pp. 228-238
[16] Bray T. *et al.*. *Extensible Markup Language (XML) 1.0 - 5th Edition. W3C recommendation, 26 Novembre 2008*. 2008 [cited November 2014]; Available from: http://www.w3.org/TR/REC-xml/.
[17] Buttler D., *A Short Survey of Document Structure Similarity Algorithms.* Proceedings of the International Conference on Internet Computing (ICOMP), 2004. pp. 3-9.
[18] Chawathe S., *Comparing Hierarchical Data in External Memory.* Inter. Conf. on Very Large Databases (VLDB), 1999, 90-101.
[19] Chawathe S. and Garcia-Molina H., *Meaningful Change Detection in Structured Data.* ACM SIGMOD, 1997. pp. 26–37.
[20] Cheriat A. *et al.*, *Incremental String Correction : Towards Correction of XML Documents.* Proceedings of the Prague Stringology Conference (PSC), 2005. pp. 201-215.
[21] Chidlovskii B., *Using Regular Tree Automata as XML Schemas.* IEEE Advances in Digital Libraries (ADL'00), 2000, 89-98.
[22] Chitic C. and Rosu D., *On Validation of XML Streams using Finite State Machines.* Proceedings of the 7th International Workshop on the Web and Databases (WebDB '04) 2004. pp. 85–90, New York, NY, USA, ACM Press.
[23] Choi B., *What are Real DTDs Like?* Proc. of the International Workshop on the Web and Databases (WebDB), 2003. pp. 43-48.
[24] Chowdhury I.J. and Nayak R., *A Novel Method for Finding Similarities between Unordered Trees Using Matrix Data Model.* International Conference on Web Information Systems and Engineering (WISE'13), 2013. pp. 421-430.
[25] Cobéna G. *et al.*, *Detecting Changes in XML Documents.* IEEE Inter. Conf. on Data Engineering (ICDE), 2002. pp. 41-52.
[26] Dalamagas T. *et al.*, *A Methodology for Clustering XML Documents by Structure.* Information Systems, 2006. 31(3):187-228.
[27] Do H., Melnik S., and Rahm E., *Comparison of Schema Matching Evaluations.* Proceedings of the International Workshop on the Web and Databases (German Informatics Society), 2002. pp. 221-237. Erfurt.
[28] Do H. and Rahm E., *Matching Large Schemas: Approaches and Evaluation.* Information Systems, 2007. 32(6): 857-885.
[29] Do H.H. and Rahm E., *Matching large schemas: Approaches and evaluation*. Information Systems, 2007. 32(6):857-885.
[30] Flesca S. *et al.*, *Detecting Structural Similarities Between XML Documents.* ACM SIGMOD WebDB, 2002. pp. 55-60.
[31] Gao S., Sperberg-McQueen C.M., and Thompson H.S. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. W3C recommendation, http://www.w3.org/TR/xmlschema11-1/ 2009 [cited May 2014].
[32] Grahne G. and Thomo A., *Approximate Reasoning in Semi-structured Databases.* Proceedings of the International Workshop on Knowledge Representation meets Databases (KRDB), 2001. Vol. 45, Rome.
[33] Helmer S., *Measuring the Structural Similarity of Semistructured Documents Using Entropy* Proceedings of the International Conference on Very Large Databases (VLDB), 2007. pp. 1022-1032.
[34] Hopcroft J. E. *et al.* , *Introduction to Automata Theory, Languages, and Computation.* 2001. Addison Wesley, 2nd edition.
[35] Huet G., *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems.* Journal of the ACM, 1980. 27:797-821.
[36] Kim S.K., Lee M., and Lee K.C., *Validation of XML Document Updates Based on XML Schema in XML Databases.* International Conference on Database and Expert Systems Applications (DEXA'03), , 2003. LNCS 2736, pp. 98–108.
[37] Klapaftis I. and Manandhar S., *Google and Wordnet based Word Sense Disambiguation.* Proceedings of the Workshop on Learning and Extending Ontologies by using Machine Learning methods 2005.

[38] Laender A. *et al., An X-ray on Web-available XML Schemas.* SIGMOD Record, 2009. 38(1): 37-42.
[39] Landau G. M and Vishkin U., *Fast Parallel and Serial Approximate String Matching.* J. of Algorithms, 1989. (10):157-169.
[40] Lee M., Yang L., Hsu W., and Yang X., *XClust: Clustering XML Schemas for Effective Integration.* Proceedings of the International Conference on Information and Knowledge Management (CIKM), 2002. pp. 292-299.
[41] Li W., L.X., Te R.,, *Cluster Dynamic XML Documents based on Frequently Changing Structures* Advances in Information Sciences and Service Sciences (AISS'12) 2012. 4(6):70-77.
[42] Liang W. and Yokota H., *SLAX: An Improved Leaf-Clustering Based Approximate XML Join Algorithm for Integrating XML Data at Subtree Classes.* Transactions of Information Processing Society of Japan, 2006. Volume 47, pp. 47-57.
[43] McGill M., *Introduction to Modern Information Retrieval.* 1983. McGraw-Hill, New York.
[44] Melnik S., Garcia-Molina H., and Rahm E., *Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching.* Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2002. pp. 117-128.
[45] Miller G., *WordNet: An On-Line Lexical Database.* International Journal of Lexicography, 1990. 3(4).
[46] Murata M. *et al., Taxonomy of XML Schema Languages Using Formal Language Theory.* ACM TOIT, 2005. 5(4):660-704.
[47] Neumann A., *Parsing and Querying XML Documents in SML.* PhD thesis, University of Trier, Trier, Germany, 2000.
[48] Nierman A. and Jagadish H. V., *Evaluating structural similarity in XML documents.* Proceedings of the ACM SIGMOD International Workshop on the Web and Databases (WebDB), 2002. pp. 61-66.
[49] Segoufin L. and Vianu V., *Validating Streaming XML Documents.* Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2002. pp. 53-64.
[50] Shoaran M. and Thomo A., *Evolving schemas for streaming XML.* Theor. Comput. Sci. , 2011. 412(35):4545-4557.
[51] Shvaiko P. and Euzenat J., *A Survey of Schema-Based Matching Approaches.* Journal of Data Semantics IV, 2005. pp. 146-171.
[52] Solimando A., Delzanno G., and Guerrini G., *Automata-based Static Analysis of XML Document Adaptation.* Third International Symposium on Games, Automata, Logics, and Formal Verification (GandALF'12), 2012. pp. 85-98.
[53] Solimando A. *et al., Static Analysis of XML Document Adaptations.* Inter. Conf. on Conceptual Modeling (ER), 2012, 57-66.
[54] Starka J. *et al.., XML Document Correction and XQuery Analysis with Analyzer.* DATESO 2011, 2011. pp. 61-72.
[55] Staworko S. and Chomicky J., *Validity-Sensitive Querying of XML Databases.* Current Trends in Database Technology – EDBT 2006, DataX'06, 2006. Lecture Notes in Computer Science, Springer, vol. 4254/2006, pp. 164–17.
[56] Su H., Padmanabhan S., and Lo M.L., *Identification of Syntactically Similar DTD Elements for Schema Matching.* Proceedings of the International Conference on Advances in Web-Age Information Management (WAIM), 2001. pp. 145-159.
[57] Suzuki N., *Finding an Optimum Edit Script between an XML Document and a DTD.* Proceedings of the ACM Symposium on Applied Computing (ACM SAC), 2005. pp. 647-653.
[58] Svoboda M. and Mlynkova I., *Correction of Invalid XML Documents with Respect to Single Type Tree Grammars.* Proceedings of NDT (Networked Digital Technologies) Communications in Computer and Information Science, 2011. 136:179-194.
[59] Taddesse F.G. *et al., Semantic-based Merging of RSS Items.* World Wide Web Journal, 2010. Vol. 12 (No. 11280), Springer.
[60] Taddesse F.G. *et al., Relating RSS News/Items.* Inter. Conf. on Web Engineering (ICWE'09), LNCS, 2009. pp. 44-452.
[61] Tansalarak N. and Claypool K. T., *QMatch - Using paths to match XML schemas.* Data Know. Eng., 2007. 60(2):260-282.
[62] Tekli J. *et al., XML Document-Grammar Comparison: Related Problems and Applications.* Central European Journal of Computer Science, 2011. Inaugural Issue, 1(1):117-136.
[63] Tekli J. *et al., Extensible User-based Grammar Matching.* Inter. Conf. on Conceptual Modeling (ER), 2009, pp. 294-314.
[64] Tekli J. *et al., Semantic and Structure Based XML Similarity: An Integrated Approach.* Inter. Conf. on Management of Data (COMAD), 2006. pp. 32-43.
[65] Tekli J. *et al., Efficient XML Structural Similarity Detection using Sub-tree Commonalities.* In Proceedings of the Brazilian Symposium on Databases (SBBD) and SIGMOD DiSC, (Best paper award), 2007. pp. 116-130.
[66] Tekli J. *et al. , A Novel XML Structure Comparison Framework based on Sub-tree Commonalities and Label Semantics.* Elsevier Journal of Web Semantics (JWS): Science, Services and Agents on the World Wide Web, 2012. 11: 14-40.
[67] Tekli J. *et al., A Fine-grained XML Structural Comparison Approach.* Inter. Conf. on Concept. Modeling (ER), 2007, 582-598.
[68] Tekli J. *et al., Structural Similarity Evaluation between XML Documents and DTDs.* Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE), 2007. pp. 196-211.
[69] Tekli J. *et al., An Overview of XML Similarity: Background, Current Trends and Future Directions.* Elsevier Computer Science Review, 2009. 3(3):151-173.
[70] Tekli J. *et al., Minimizing User Effort in XML Grammar Matching.* Elsevier Information Sciences Journal, 2012. 210:1-40.
[71] Tekli J. *et al., Differential SOAP Multicasting.* IEEE Inter. Conf. on Web Services (ICWS'11), 2011. pp. 1-8, Washington DC.
[72] Tekli J. *et al., Using XML-based Multicasting to Improve Web Service Scalability.* Inter. J. on Web Services Research (IJWSR), 2012. 9(1):1-29.
[73] Tekli J. *et al., Approximate XML Structure Validation.* Technical Report ApproXMLVal-TR-14, LAU-ICMC-LIUPPA,ICMC, 2014, http://sigappfr.acm.org/Projects/XS3/ApproXMLVal-TR-14.pdf
[74] Teraguchi M. *et al., Optimized Web Services Security Performance with Differential Parsing.* Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC'06), 2006. pp. 277-288.
[75] Thomo A. *et al., Visibly Pushdown Transducers for Approximate Validation of Streaming XML.* International Symposium on Foundations of Information and Knowledge Systems (FoIKS), 2008. pp. 219-238.
[76] Toyama Y., *On the Church-Rosser Property for the Direct Sum of Term Rewriting Systems.* J. of the ACM, 1987. 34:128-143.
[77] W3 Consortium. *The Document Object Model.* 2005 [cited 28 May 2009]; Available from: http://www.w3.org/DOM.
[78] Werner C. *et al., WSDL-Driven SOAP Compression.* Inter. Journal of Web Services Research, 2005. (2)1:18-35.
[79] Xing G., *Fast Approximate Matching Between XML Documents and Schemata.* The Asia Pacific Web Conf., 2006. pp. 425-436.
[80] Xing G. *et al., Computing Edit Distances Between an XML Document and a Schema and its Application in Document Classification.* Proceedings of SAC 06, Dijon, France, 2006. pp. 831-835. ACM.
[81] Zhang K. *et al., Approximate Tree Matching in the Presence of Variable Length Don't Cares.* Journal of Algorithms, 1994. (16) 33-66.
[82] Zhang K. and Shasha D., *Simple Fast Algorithms for the Editing Distance between Trees and Related Problems.* SIAM Journal of Computing, 1989. 18(6):1245-1262.
[83] Zhang Z. *et al. , Similarity Metric in XML Documents.* Knowledge Management and Experience Management Workshop, 2003.
[84] Zirn C. *et al., Distinguishing between Instances and Classes in the Wikipedia Taxonomy.* European Semantic Web Conference (ESWC), 2008. 376-387.

Appendixes

Appendix I. Mathematical Analysis regarding the Disjunctive Normal Form (*DNF*)

In the following, we consider three common configurations of '*Or*' operator expressions: i) concatenated, ii) encapsulated, and iii) mixed; which usually appear in real XML grammars (based on empirical analyses in [11, 23, 38], as well as the grammars utilized in our own experiments, described in Section 5 of the main paper). In the following, we show on one hand that with concatenated and/or encapsulated configurations, the number of conjunctive declarations resulting from the *DNF* representation of a grammar expression is linear in the number of '*Or*' operators involved, denoted *Nb*(*Or*). On the other hand, we show that mixed declarations might induce, in certain specific cases, an exponential increase in the number of conjunctive grammars.

I.1. Grammars with Concatenated 'Or' expressions

These correspond to grammars containing alternative expressions where *'Or'* operators appear exclusively at the same level within the same grammar expression: <!ELEMENT $A(B|C|D|...)$> such as *'A'* is the root node, or an inner node in the grammar, and *'B', 'C', 'D',* ... are either: i) single node declaration, ii) empty node declarations, or iii) sequence expressions (i.e., expressions of elements connected via the *'And'* operators). Given an XML grammar *G* consisting solely of concatenated '*Or*' expressions (i.e., yielding the maximum *Nb*(*Or*) possible following this configuration), and based on the inductive mathematical reasoning in Table I, the maximum number of conjunctive grammars N_G resulting from the *DNF* representation of *G*, $DNF(G) = \{C\}_G$, is equal to:

$$N_G = Nb(Or) + 1 \quad \text{such as} \quad N_G = |G| - 1 \text{ and } Nb(Or) = |G|-2 \qquad \textbf{(i)}$$

Table I. XML Grammars made of concatenated *'Or'* operators, such as the number of *'Or'* operators, and consequently the number of conjunctive grammars N_G, are maximized.

Grammar expressions	*Nb*(*Or*)	$\|G\|$	N_G	$\|C_G\|$
<ELEMENT root (a | b)>	1	3	2	2
<ELEMENT root (a | b | c)>	2	4	3	2
<ELEMENT root(a | b | c | d)>	3	5	4	2
<ELEMENT root(a | b | c | d | e)>	4	6	5	2
<ELEMENT root(a | b | c | d | e | f)>	5	7	6	2
...				
Recursively,	**|G|-2**	**|G|**	**|G|-1**	**2**

I.2. Grammars with Encapsulated 'Or' expressions

These are grammars containing alternative expressions where *'Or'* operators are encapsulated in each other, such as no two *'Or'* operators appear at the same structural level. Such grammars are of the form: <!ELEMENT $A(B|C)$> <!ELEMENT $B(E|F)$> <! ELEMENT $E(H|I)$> ... where *'A'* is the root node or an inner node, and '*B', 'C', 'E',* ... are either i) single node declaration, ii) empty node declarations, or iii) sequence expressions (i.e., expressions of elements connected via the *'And'* operators). Thus, given an XML grammar *G* consisting solely of encapsulated '*Or*' expressions (i.e., yielding the maximum *Nb*(*Or*) possible following this configuration), and based on the inductive mathematical reasoning in Table II, the maximum number of conjunctive grammars N_G resulting from $DNF(G) = \{C\}_G$, is equal to:

$$N_G = \text{Nb}(Or) + 1 \quad \text{such as} \quad N_G = \frac{|G|+1}{2} \quad \text{and} \quad \text{Nb}(Or) = \frac{|G| - 1}{2}$$

Table II. Grammars made of encapsulated *'Or'* operators, such as the number of '*Or'* operators, and consequently the number of conjunctive grammars N_G, are maximized.

Grammar expressions	*Nb*(*Or*)	$\|G\|$	N_G	$\| C_G \|$
<ELEMENT root (a | b)>	1	3	2	2
<ELEMENT root (a | b)> *<ELEMENT a (c | d)>*	2	5	3	3
<ELEMENT root(a | b)> *<ELEMENT b (c | d)>* *<ELEMENT c (e | f)>*	3	7	4	4
<ELEMENT root(a | b)> *<ELEMENT b (c | d)>* *<ELEMENT c (e | f)>* *<ELEMENT e (g | h)>*	4	9	5	5
<ELEMENT root(a | b)> *<ELEMENT b (c | d)>* *<ELEMENT c (e | f)>* *<ELEMENT e (g | h)>* *<ELEMENT g (h | i)>*	5	11	6	6
...				
Recursively,	$\frac{\|G\|-1}{2}$	$\|G\|$	$\frac{\|G\|+1}{2}$	$\frac{\|G\|+1}{2}$

I.3. Grammars with mixed expressions

These are grammars made of expressions containing both sequence (*And*) and alternative (*Or*) expressions. Here, 2 main configurations can occur, which we identify as: i) *And-Or* expressions, and ii) *Or-And* expressions.

- ***And-Or*** expressions – These are of the form $E = (A_1 \mid A_2 \mid \ldots \mid A_n)$ where each B_i consists of a sequence expression, denoted $B_i = (B_1, B_2, \ldots, B_m)$. These come down to the cases of concatenated and/or encapsulated '*Or*' expressions described above, since sequence expressions are not affected via the *DNF* representation and can be processed as single node declarations.
- ***Or-And*** expressions – these are of the form $E = (A_1, A_2, \ldots, A_n)$ where each A_i consists of an alternative expression, denoted $A_i = (B_1 \mid B_2 \mid \ldots \mid B_m)$. Here, one can realize that the number of conjunctive expressions resulting from the *DNF* representation of E is exponential in the number of A_i expressions in E (i.e., the cardinality of E w.r.t. the main '*And*' sequence operator):

$$N_E = |A_i|^{|E|} = m^n \quad \text{such as} \quad Nb(Or) = |E| \times (|A_i|-1) = n \times (m-1)$$

Consequently, a grammar G containing multiple mixed expressions E_i, transformed into its *DNF* representation, would inherently yield an exponential increase in the number of conjunctive grammars N_G (in comparison with the linear dependencies described in the concatenated and encapsulated cases discussed above).

To summarize, the number of conjunctive grammars N_G resulting from the *DNF* representation of an XML grammar G remains linear in the number of '*Or*' operators involved in G, in many practical cases including: concatenated '*Or*' expressions, encapsulated '*Or*' expressions, and mixed *And-Or* expressions. However, an exponential increase in N_G can occur in the case of mixed *Or-And* expressions**.** Here, note that mixed expressions (including both *And-Or* and *Or-And* declarations) only cover 12.65% of grammar expressions used in real XML grammars [11, 23, 38].

Appendix II. XML *Grammar_to_Tree* Algorithm

The pseudo-code of our algorithm for transforming an XML grammar into its tree representation, entitled *XGram_to_Tree*, is shown in Fig. I. Given an XML grammar G, an XML document tree D to be compared with the grammar, and our set of transformation rules $\{R\}$ (cf. Table 1 and Table 2), the algorithm first runs the grammar G through the general transformation rules (rules 1 to 3, cf. Table 1) so as to flatten repeatable expressions, eliminating all cardinality constraints associated to the *And* and *Or* operators (cf. Fig. I, line 1, the rules being recursively applied on the input grammar G until no further transformations are possible). Then, the algorithm runs the resulting (flattened) grammar G_1 though the *one-to-one* transformation rules (rules 2+, 3+, and 4, cf. Table 2), handling *MaxOccurs='unbounded'* and *recursive* expressions w.r.t. the XML document tree at hand (line 2). Subsequently, the algorithm transforms the resulting grammar G_2 into its *disjunctive normal form*, $DNF(G_2) = \{C\}_{G2}$ (line 3) and then represents each resulting conjunctive XML grammar as a single *rooted ordered labeled tree* (lines 4-8).

```
Algorithm XGram_to_Tree
Input: G                    // XML grammar
       D                    // XML Document tree (to be compared with G )
       {R}                  // Set of transformation rules, cf. Table 1 and Table 2
Output: Gram_TreeSet        // Set of conjunctive grammar trees representing G
Begin
 Repeat G --Ri--> G1  i = 1, 2, 3   While (G1 ≠ G)          // Applying general transformation rules in Table 1[1]  1
 Repeat G1 --Rj(D)--> G2  j = 2+, 3+, 4   While (G2 ≠ G1)   // Applying one-to-one transformation rules in Table 2[1]  2
 {C}G = DNF(G2)             // Transforming the simplified grammar into its Disjunctive Normal Form   3
  For each Ci ∈ {C}G                                                                                  4
  {                                                                                                   5
     Ci_Tree = Conjunctive_Gram_Tree_Representation(Ci)   // following our tree model, cf. Definition 16  6
     Gram_TreeSet U Ci_Tree                                                                           7
  }                                                                                                   8
  Return Gram_TreeSet       // Set of conjunctive grammar trees representing G                        9
End
```

Fig. I. Pseudo-code of *XGram_to_Tree*, for transforming an XML grammar into its tree representation.

[1] Recall that the rule application order is irrelevant since the rules follow the *Extended Church-Rosser* (*ECR*) property.

Appendix III. Definitions of Edit Operations used in our TED_{XDoc_XGram} Approach

Definition 1 - Insert Leaf Node: Let T be a tree with a node $p \in N_T$, and let $T_1, \ldots, T_m$ be the first level sub-trees corresponding to node p (i.e., sub-trees rooted at the children of node p). Given a node x not belonging to T, $x \notin T$, $Ins(x, i, p, \ell)$ is a node insertion operation applied to T, inserting x as the ith leaf child of p. In the transformed tree T', node p will have $T_1, \ldots, T_{i-1}, x, T_{i+1}, \ldots, T_{m+1}$ as its first level sub-trees, with ℓ the label of inserted leaf node x ●

Definition 2 – Delete Leaf Node: Given a leaf node x in tree T, i.e., $x \in N_T$ such as $x.Deg = 0$, $Del(x)$ is a node deletion operation applied to T, yielding T' where node p will have first level sub-trees $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_m$ ●

Definition 3 – Update Node (Label): Given a node x in tree T, $x \in N_T$, and a label ℓ, $Upd(x, \ell)$ is a node (label) update operation applied to x resulting in T' which is identical to T except that in T', x bears ℓ as its label. The update operation could be also formulated as follows: $Upd(x, y)$ where $y.\ell$ denotes the new label to be assumed by x ●

Definition 4 – Insert Tree: Let T be a tree, with a node $p \in N_T$, and let $T_1, \ldots, T_m$ be the first level sub-trees of node p. Given a tree A not belonging to T, $InsTree(A, i, p)$ is a tree insertion operation applied to T, inserting A as the ith sub-tree of p. In the transformed tree T', node p will have $T_1, \ldots, T_{i-1}, A, T_{i+1}, \ldots, T_{m+1}$ as its first level sub-trees ●

Definition 5 – Delete Tree: Let T be a tree with a node $p \in N_T$, having a tree A as the ith first level sub-tree of p, $DelTree(A)$ is a tree deletion operation applied to T, yielding T' where node p will have first level sub-trees $T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_m$ ●

Appendix IV. Properties of our XML Document/Grammar Similarity Measure

IV.1. Tree Edit Distance Minimality Property

Given an XML tree D and a conjunctive grammar tree C, one can transform D into a document tree conforming to C using any of an infinite number of edit scripts (e.g., repeatedly inserting and deleting the same node and/or sub-tree, etc.). Such edit scripts are obviously meaningless in the context of our study since we aim to identify the *minimum cost edit script:* that applies the fewer and minimum cost operations transforming D into a document tree valid w.r.t. C. In other words, if we consider $L(C)$ to be the set of document trees generated by grammar C (i.e., the language of grammar C), then we aim to identifying the minimum distance (cost) necessary to transform document tree D into any document $D' \in L(C)$. Hence, the distance minimality property carries immediately from our definition of tree edit distance (cf. TED_{XDoc_XGram} in Fig. 12) and serves as the main directive in defining our TED_{XDoc_XGram} algorithm.

IV.2. Similarity Measure Metric Properties

Our similarity measure in formula (5) is consistent with the formal definition of similarity, as a (semi-) metric function satisfying (in part) the metric properties of *Reflexivity*, *Minimality*, *Symmetricity* and *Triangular Inequality*. Here, note that while XML documents and XML grammars are different in nature (i.e., XML documents underline data instances, whereas XML grammars underline data type), yet an XML document tree, following our model, comes down to a conjunctive XML grammar tree free of cardinality constraints operators (i.e, when all elements of the grammar tree are associated default constraints *MinOccurs* = *MaxOccurs* = 1, which are equivalent to *null* constraints and can be omitted). This allowed us to verify the following metric properties:

i. $Sim_{XDoc\_XGram}(D, G) \in [0, 1]$[1].
ii. $Sim_{XDoc\_XGram}(D, G) = 1 \Rightarrow$ XML document tree D conforms to grammar G ($G \vDash D$).
iii. $Sim_{XDoc\_XGram}(D, D) = 1 \Rightarrow$ Reflexivity.
iv. $Sim_{XDoc\_XGram}(D_1, D_2) \leq Sim_{XDoc\_XGram}(D, D) \Rightarrow$ Minimality.
v. $Sim_{XDoc\_XGram}(D_1, D_2) = Sim_{XDoc\_XGram}(D_2, D_1) \Rightarrow$ Symmetricity.
vi. $Sim_{XDoc\_XGram}(D_1, D_3) \geq Sim_{XDoc\_XGram}(D_1, D_2) \times Sim_{XDoc\_XGram}(D_2, D_3) \Rightarrow$ Triangular inequality, (i.e., $TED_{XDoc\_XGram}(D_1, D_3) \leq TED_{XDoc\_XGram}(D_1, D_2) + TED_{XDoc\_XGram}(D_2, D_3)$).

Note that our measure is a semi-metric (and not a *full* metric) since: i) it does not allow comparing two grammars (i.e., $Sim(G_1, G_2)$), nor ii) using a grammar as the first parameter of the similarity measure ($Sim(G, D)$ is not allowed, i.e., we cannot transform grammar G so that it includes in its language document D. We do it the other way around: transforming D so that it becomes $D \vDash C$). Comparing/transforming grammars is out of the scope of this study.

---

[1] In practice, we will hardly ever obtain $TED_{XDoc\_XGram} = \infty$. Hence, Sim_{XDoc_XGram} values will hardly ever reach 0. Note that alternative similarity formulas, such as $Sim_{XDoc\_XGram}(D, G) = 1 - (TED_{XDoc\_XGram}(D, G) / (|D| + |G|))$ could be used to bring similarity values to a limited range, where $Sim_{XDoc\_XGram} = 0$ is more practically attainable. Yet, such a formula would violate the *triangular inequality* metric property, which is why it is disregarded in this approach.

Appendix V. Detailed Computation Examples

V.1. Extended TED Recurrence (TED$^+$) Computations

Consider the example in Fig. 14. Fig. 14.a depicts the computation of *Dist*[1, 1] between partial document tree *E<1>* and partial grammar tree *C<1>* and has been described in detail in the main paper (in short, no changes need to be made to *E<1>* since *C<1>* $\vDash$ *E<1>*). Fig. 14.b depicts the computation of *Dist*[1, 2] between partial document tree *E<1>* and *C<2>*. Computing the α factor consists of deleting sub-tree E_1 (consisting of leaf node *a*), which cost =1. Computing the β factor consists in inserting 2 occurrences of sub-tree C_2, in order to fulfill the corresponding $R(C_2).MinOccurs$ = 2 constraint in the transformed partial document tree *E<1>* to obtain *C<2>* $\vDash$ *E<1>'*. Computing the γ factor consists in evaluating the edit distance between sub-tree E_2, the (only existing) match candidate with grammar sub-tree C_2 (*NbOcc*[2] = 1). Nonetheless, *NbOcc*[2] < $R(C_2).MinOccurs$=2 shows that one more occurrence of C_2 is required to appear in *E<1>* to obtain *C<2>* $\vDash$ *E<1>'*. Thus, γ_1 is applied to account for the *remaining sub-tree occurrence*, yielding γ = 3, which is the cost of inserting an occurrence of C_1 into *E<1>*. Consequently, *Dist*[1, 2] = *Min*(α, β, γ) = γ = 3, indicating that the minimum (cost) amount of change required to transform *E<1>* in order to obtain *C<2>* $\vDash$ *E<1>'* is to insert an additional occurrence of C_2 in *E<1>*.

Similarly, consider Fig. 14.c which depicts the computation of *Dist*[4, 2] between partial document tree *F<4>* and partial grammar tree *C<2>*. Computing the α factor consists in deleting sub-tree F_4, which cost = 2. Computing the β consists in inserting 2 occurrences of sub-tree C_2, in order to fulfill the corresponding $R(C_2).MinOccurs$ = 2 constraint. Computing the γ factor consists in evaluating the edit distance between document sub-trees F_2, F_3, F_4 on one hand, which are the consecutive first-level sub-trees in *F<4>* which could match C_2, and grammar sub-tree C_2 on the other hand. Here, *NbOcc*[2]=3 $\in$ [$R(C_1).MinOccurs$=2, $R(C_1).MaxOccurs$=3], thus γ_2 is applied. This yields cost = 1, indicating that $C_2 \vDash \{F_2, F_3\}$ (inducing no edit distance cost, $TED_{XDoc\_XGram}(F_2, C_2) + TED_{XDoc\_XGram}(F_3, C_2) = 0$), while $C_2 \mid\!\approx F_4$, requiring the inserting of node *e* ($TED_{XDoc\_XGram}(F_4, C_2) = \text{Cost}_{\text{InsTree}}(C_{22}) = 1$) to obtain $C_2 \vDash F_4'$. Then, *Dist*[4, 2] = *Min*(α, β, γ) = γ = 1, indicating that the minimum (cost) amount of change required to transform *F<4>* in order to obtain *C<2>* $\vDash$ *F<4>'*, is to insert node *e* under sub-tree F_4.

V.2. Complete TED_{XDoc_Gram} Matrix Computations

Consider the edit distance matrixes in Fig. 15, depicting all recurrences when running the TED_{XDoc_Gram} algorithm to compare document trees *D*, *E*, *F* with grammar *C* of Fig. 13.

Fig. 14.a depicts the computation of $TED_{Xdoc\_Xgram}(D, C)$ and has been discussed in detail in the main paper (in short, $TED_{Xdoc\_Xgram}(D, C) = 0 \Rightarrow Sim_{Xdoc\_XGram}(D, C) = 1 / (1+TED_{XDoc\_XGram}(D, C)) = 1 \Rightarrow C \vDash D$).

Fig. 14.b depicts $TED_{XDoc\_XGram}(E, C) = 3 \Rightarrow \text{Sim}_{\text{Xdoc\_XGram}}(E, C) = 1 / (1 + \text{TED}_{\text{XDoc\_XGram}}(E, C)) = 0.25 \Rightarrow C \mid\!\approx_{0.25} E$, i.e., document tree *E* approximately validates *C* with a similarity score = 0.25. Here, *Dist*[0, 0] = 0 since the document and grammar tree roots match. Dist[1, 1] = Dist[0, 0] + $TED_{XDoc\_XGram}(E_1, C_1) \equiv \text{Cost}_{\text{Upd}}(R(E_1).\ell, R(C_1).\ell)$ = 0 since $C_1 \vDash D_1$. *Dist*[2, 2] = *Dist*[1, 1] + $TED_{XDoc\_XGram}(E_2, C_2) + \text{Cost}_{\text{InsTree}}(C_2) \times (R(C_2).MinOccurs - NbOcc[2])$ = 0 + 0 + 3 = 3, since D_2 is the only (exact) match of C_2, $C_2 \vDash D_2$, (*NbOcc*[2] = 1) whereas the minimum number of occurrences of C_2 required to appear in the document tree is $R(C_2).MinOccurs$ = 2 (thus we need to consider the cost of inserting an additional occurrence of C_2, i.e., $\text{Cost}_{\text{InsTree}}(C_2)$ = 3, in order to obtain *C<2>* $\vDash$ *D<2>'*). *Dist*[3, 3] = *Dist*[2, 2] + $TED_{XDoc\_XGram}(E_3, C_3) = \text{Cost}_{\text{Upd}}(R(E_3).\ell, R(C_3).\ell)$ = 3 + 0 since $C_3 \vDash E_3$ (given that one occurrence of C_3 is required, and has actually appeared in the document tree). To sum up, $TED_{XDoc\_XGram}(E, C) = 3 \Rightarrow C \mid\!\approx_{0.25} E$ highlights the cost of inserting one additional occurrence of sub-tree C_2 into document tree *E*, to obtain $C \vDash E'$.

Fig. 14.c depicts $TED_{XDoc\_XGram}(F, C) = 4 \Rightarrow Sim_{XDoc\_XGram}(F, C) = 1 / (1 + TED_{XDoc\_XGram}(F, C)) = 0.2 \Rightarrow C \mid\!\gg_{0.2} E$. Here, *Dist*[0, 0] = 0 since the document and grammar tree roots match. Dist[1, 1] = Dist[0, 0] + $TED_{XDoc\_XGram}(F_1, C_1) \equiv \text{Cost}_{\text{Upd}}(R(F_1).\ell, R(C_1).\ell)$ = 0 since $C_1 \vDash F_1$. *Dist*[4, 2] = *Dist*[1, 1] + $TED_{XDoc\_XGram}(F_2, C_2)$ + $TED_{XDoc\_XGram}(F_3, C_2)$ + $TED_{XDoc\_XGram}(F_4, C_2)$ = 0 + 0 + 0 + 1 = 1 since $C_2 \vDash \{F_2, F_3\}$ whereas $C_2 \mid\!\approx F_4$ requiring the insertion of node *e* ($TED_{XDoc\_XGram}(F_4, C_2) = \text{Cost}_{\text{InsTree}}(C_{22}) = 1$) in order to obtain $C_2 \vDash F_4$ (having NbOcc[2]=$R(C_3).MaxOccurs$ = 3, i.e., 3 occurrences of C_2 are allowed to appear, and have actually appeared in the document tree). *Dist*[5, 2] = *Dist*[4, 2] + $Cost_{DelTree}[5]$ = 1 + 3 = 4, considering the cost of deleting sub-tree F_5, since F_5 is considering as an additional yet unwanted occurrence of sub-tree C_2 (*NbOcc*[2]=4 > $R(C_3).MaxOccurs$=3). *Dist*[6, 3] = *Dist*[5, 2] + $TED_{XDoc\_XGram}(F_6, C_3)$ = 4, where $TED_{XDoc\_XGram}(F_6, C_3) \equiv \text{Cost}_{\text{Upd}}(R(F_6).\ell, R(C_3).\ell)$ = 0 since $C_3 \vDash F_6$. To sum up, $TED_{XDoc\_XGram}(F, C) = 4 \Rightarrow C \mid\!\gg_{0.2} E$ underlines the costs of i) inserting node *e* in sub-tree F_4, and ii) deleting sub-tree F_5 from document tree *F*, in order to obtain $C \vDash F'$. This means that *F* requires more costly transformations conform to grammar tree *C* in comparison with document tree *E*.

In addition, consider a simple variation of grammar *C* where sub-tree C'_2's root node is assigned $R(C_2).MaxOccurs = \infty$ instead of *MaxOccurs* = 3 (all other nodes remaining the same). In this case, the edit distance

table in Fig. 14.c would remain the same except for: *Dist*[5, 2] = *Dist*[4, 2] + $TED_{XDoc\_XGram}(F_5, C_2)$ = 1 + 1 = 2, where: *Dist*[4, 2] = 1 underlines the cost of inserting node *e* under sub-tree F_4 in order to obtain $C_2 \vDash F_4'$ (similarly to the previous example), and $TED_{XDoc\_XGram}(F_5, C_2) \equiv \text{Cost}_{\text{Upd}}(R(F_{52}).\ell, R(C_{22}).\ell) = 1$, transforming node label *f* into *e* in sub-tree F_5 ($TED_{XDoc\_XGram}(F_5, C_2) = \text{Cost}_{\text{Upd}}(R(\text{F}_{52}).\ell, \text{R}(\text{C}_{22}).\ell,) = 1$) in order to obtain $C_2 \vDash F_5'$. In other words, F_5 is now considered as an (approximate) occurrence of sub-tree C_2, since an infinite number of occurrences of C_2 is accepted in the grammar tree (NbOcc[2]=4 < $R(C_2)$.MaxOccurs = ∞), in comparison with *Example 3* where F_5 was an unwanted occurrence (to be deleted from *F* in order to obtain $C \vDash F'$). To sum up, having $R(C_2).MaxOccurs = \infty$, $TED_{XDoc\_XGram}(F, C) = 2 \Rightarrow C \mid\approx_{0.334} F$, designating the costs of i) inserting node *e* under sub-tree F_4, and ii) updating node label *f* into *e* in sub-tree F_5 (having $C_2 \mid\approx \{F_4, F_5\}$) in order to obtain $C \vDash F'$.

V.3. Edit Distance Matrixes concerning the Running Example (*in Section* 4.4.3).

Table III. Tree edit distance computations when comparing XML document tree *D* and XML conjunctive grammar tree C_{I}.[1]

	R(C_{I}) (*Paper*)	$C_{\text{I}1}$ (*Category, MinOccurs=0*)	$C_{\text{I}2}$ (*Title*)	$C_{\text{I}3}$ (sub-tree of root *Author, MinOccors=2 MaxOccurs=10*)	$C_{\text{I}4}$ (*Version*)	$C_{\text{I}5}$ (*Length, MinOccurs=0*)	$C_{\text{I}6}$ (sub-tree of root *url, MinOccurs=0 MaxOccurs=∞*)
R(D) (*Paper*)	0	0	1	9	10	10	10
D_1 (*Title*)	1	1	0	8	9	9	9
D_2 (sub-tree of root *Publisher*)	4	4	3	4	5	5	5
D_3 (*Version*)	5	5	4	5	4	4	4
D_4 (*Length*)	6	6	5	6	5	4	4
D_5 (sub-tree of root *url*)	12	12	11	12	11	10	4

Table IV. Tree edit distance computations when comparing XML document tree *D* and XML conjunctive grammar tree C_{II}.

	R(C_{II}) (*Paper*)	$C_{\text{II}1}$ (*Category, MinOccurs=0*)	$C_{\text{II}2}$ (*Title*)	$C_{\text{II}3}$ (*Publisher*)	$C_{\text{II}4}$ (*Version*)	$C_{\text{II}5}$ (*Length, MinOccurs=0*)	$C_{\text{II}6}$ (sub-tree of root *url, MinOccurs=0 MaxOccurs=∞*)
R(D) (*Paper*)	0	0	1	2	3	3	3
D_1 (*Title*)	1	1	0	1	2	2	2
D_2 (sub-tree of root *Publisher*)	4	4	3	2	3	3	3
D_3 (*Version*)	5	5	4	3	2	2	2
D_4 (*Length*)	6	6	5	4	3	2	2
D_5 (sub-tree of root *url*)	12	12	11	10	9	8	2

Table V. Tree edit distance computations when comparing XML document tree *D* and XML conjunctive grammar tree C_{III}.

	R(C_{III}) (*Paper*)	$C_{\text{III}1}$ (*Category, MinOccurs=0*)	$C_{\text{III}2}$ (*Title*)	$C_{\text{III}3}$ (sub-tree of root *Publisher*)	$C_{\text{III}4}$ (*Version*)	$C_{\text{III}5}$ (*Length, MinOccurs=0*)	$C_{\text{III}6}$ (sub-tree of root *url, MinOccurs=0 MaxOccurs=∞*)
R(D) (*Paper*)	0	0	1	2	3	3	3
D_1 (*Title*)	1	1	0	1	2	2	2
D_2 (sub-tree of root *Publisher*)	4	4	3	0	1	1	1
D_3 (*Version*)	5	5	4	1	0	1	1
D_4 (*Length*)	6	6	5	2	1	0	0
D_5 (sub-tree of root *url*)	12	12	11	8	7	6	0

[1] Recall that *MinOccurs* = 1 and *MaxOccurs* = 1 designate default values which are equivalent to *null* constraints, and thus can be omitted in the edit distance matrixes (for ease of presentation).