Measuring the Structural Similarity among XML Documents and DTDs

E. Bertino

Dipartimento di Scienze dell'Informazione

Università degli Studi di Milano

Via Comelico 39/41 - I20135 Milano, Italy

bertino@dsi.unimi.it

G. Guerrini *M. Mesiti* *I. Rivara* *C. Tavella*

Dipartimento di Informatica e Scienze dell'Informazione

Università degli Studi di Genova

Via Dodecaneso 35 - I16146 Genova, Italy

{guerrini,mesiti}@disi.unige.it

Abstract

Sources of XML documents are proliferating on the Web and documents are more and more frequently exchanged among sources. At the same time, there is an increasing need of exploiting database tools to manage this kind of data. An important novelty of XML is that information on document structures is available on the Web together with the document contents. This information can be used to improve document handling and to achieve more effective and efficient searches on documents. However, in such an heterogeneous environment as the Web, it is not reasonable to assume that XML documents that enter a source always conform to a predefined DTD present in the source.

In this paper we propose a metric for quantifying the structural similarity between an XML document and a DTD. This metric can be employed for document classification and document clustering. In the first case, the proposed metric is used for selecting the DTD, among the ones in the source, whose structure is the most similar to the structure of the document. In the second case, the aim is to group together documents accordingly to the similarity of their structures. The documents, whose similarity to a given DTD is high enough, can then be grouped in the structural cluster corresponding to that DTD.

1 Introduction

XML [17] has recently emerged as the most relevant standardization effort for document representation and exchange on the Web. It offers the possibility of defining tags and modeling nested document structures. XML documents, thanks to semantic tags, are self-describing. Thus, with the advent of XML, information on document structures (provided through the tags embedded in a document) is now available on the Web. The knowledge of a document structure can obviously be used in order to improve document retrieval. For instance, knowing that a given term, say *daisy*, appears as value of a `first-name` element rather than of a `flower` element, allows one to determine whether or not the term is meaningful for a given query.

XML also offers the possibility of defining document types (called DTDs -*Document Type Definitions*) that describe the structure of documents. An XML DTD is essentially a grammar constraining the tags and the structure of the document. A document whose structure conforms to a DTD is called *valid* in XML terminology. Obviously, the presence of a DTD allows one to take advantage of the knowledge about document structures for storing, querying, and indexing a set of documents.

In this paper, we refer to XML both as a data

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

We thus address the problem of matching documents against a set of DTDs even when such documents do not fully conform to any DTD in such set. Specifically, we allow some attributes and subelements specified for an element in the DTD to be missing in the corresponding element of the document, and, viceversa, we allow the document to contain some additional attributes and subelements not appearing in the DTD. Moreover, we allow elements/attributes in the document to follow a different order w.r.t. the one specified in the DTD (i.e. we are focusing on data-centric documents [5]). Finally, the tags in the document and in the DTD could not be exactly the same, provided they are *stems* or are similar enough according to a given Thesaurus. In matching a document against a DTD the goal is then to quantify, through an appropriate measure, the similarity between the document and the DTD.

The obtained measure can be employed both for document classification and for document clustering. In the case of document classification, the scenario we refer to consists of a number of heterogeneous sources of XML documents able to exchange documents among each other. Each source stores and indexes the local documents according to a set of local DTDs. An XML document entering a source is matched against the DTDs in the source. If a DTD exists in the source to which the document conforms according to the usual notion, then the document is accepted as valid for this DTD. Otherwise, the proposed metric is used for selecting the DTD, among the ones in the source, that best describes the structure of the document. The approach can also be used to support a *structural clustering* of documents, based on the idea of grouping together documents accordingly to the similarity of their structures. All the documents, whose similarity to a given DTD is high enough, will be grouped in the structural cluster corresponding to that DTD. As clustering [14] assembles together documents with similar terms, structural clustering assembles together documents with a similar structure. Moreover, if we express a user query as a DTD, structural clustering can be used to find a better formulation of the query, like in the case of content-based clustering.

In the case of document clustering, again we start from a set of DTDs. These DTDs can be predefined, can be derived from a user query, or can even be inferred from a sample set of documents

through existing DTD inference techniques, like [8, 13]. An XML document is matched against the DTDs in the set. If a DTD exists in the set to which the document conforms according to the usual notion, or to which the document structure is similar enough, the document is assigned to the structural cluster corresponding to this DTD.

The matching process is performed against a tree representation of both documents and DTDs. This representation can be easily obtained from their declarations. Though our technique handles all the features of XML documents, in the paper, for the sake of clarity, we will focus on the most meaningful *core* of the approach, thus we restrict ourselves to a subset of XML documents and to tag equality. We will then discuss how the approach generalizes to arbitrary XML documents and how it is able to handle tag similarity.

Our proposal is, to the best of our knowledge, the first one addressing the issue of measuring the structural similarity of XML documents and DTDs, and it has relevant potential applications for structural clustering of documents and for document retrieval based on conditions on the document structure. Our work benefits from previous work developed in the context of automatic object classification [3] and of heterogeneous information integration (both at the data [12] and at the schema [7] levels). Ours is, however, the first approach developed for XML, which takes into account both tag and structure similarities.

The remainder of the paper is structured as follows. Section 2 presents our tree representation of XML documents and DTDs. Section 3 discusses the basic ideas to evaluate the structural similarity between a document and a DTD, whereas in Section 4 the similarity function and the matching process are discussed in details. Section 5 discusses extensions and applicability of the proposed technique, and presents experimental results. Finally, Section 6 concludes the work. Appendix A contains the matching algorithm.

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

Figure 1: An example of XML document

2 Documents and DTDs as Trees

Figure 1 shows an example of XML document describing software products whereas Figure 2 shows a possible corresponding DTD. The example shows that, in a DTD, for each subelement it is possible to specify whether it is optional ('?'), whether it may occur several times ('*' for 0 or more times, and '+' for 1 or more times), and whether some subelements are alternative with respect to each other ('|'). As we have anticipated in the introduction, we will focus on a subset of XML documents. In particular, we will only consider elements (that can have a nested structure)

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

Figure 2: An example of DTD

disregarding attributes (that can be seen as a particular case of elements). Since we disregard attributes, we will only consider nonempty elements. Moreover, since we are interested in data-centric documents, we disregard the order of subelements. In the matching process, we represent both DTDs and XML documents through labeled trees. In this section we discuss this representation.

2.1 Tree Representation of Documents

An XML document is represented as a labeled tree. This representation only relies on information determined from the structure of the document. Our definition is based on the classical definition of labeled tree. That is, given a set $\mathcal{N}$ of nodes and a set $\mathcal{A}$ of labels, a labeled tree is defined by induction as follows: $v \in \mathcal{N}$ is a labeled tree; if $T_1, \ldots, T_n$ are labeled trees, $a_1, \ldots, a_n \in \mathcal{A}$, and $v \in \mathcal{N}$, then $(v, [(a_1, T_1), \ldots, (a_n, T_n)])$ is a labeled tree.

In our representation of documents each node represent an element or a value. The label associated with the edge incoming to a node represents the corresponding tag name or value. The labels used to label the tree belong to a set of element tags ($\mathcal{EN}$) and to a set of values that the data content of an element can assume ($\mathcal{V}$). In each tree that represents an XML document there is a single edge outgoing from the root, and the label of this edge belongs to $\mathcal{EN}$ (it is the name of the document element). Moreover, all edges labeled by values in $\mathcal{V}$ are incoming to nodes that are leaves of the tree (and are called *terminal edges*). The following definition introduces our representation of XML documents.

Definition 1 (*XML document*). Let $\mathcal{EN}$ be a set of element tags and $\mathcal{V}$ be a set of values. An XML document D is a labeled tree on $\mathcal{EN} \cup \mathcal{V}$ with the following properties:

1. $D = (v, [(l, T')])$ with $l \in \mathcal{EN}$;
2. for each $T = (v, C)$ subtree of D, if l is the label of the edge incoming to T, then $l \in \mathcal{EN}$;
3. for each $T = v$ (leaf) subtree of D, if l is the label of the edge incoming to T, then $l \in \mathcal{V}$. □

For the sake of simplicity, in the graphical representation we omit the explicit direction of edges. All edges are oriented downward. Figure 3 shows the tree representation of the XML document presented in Figure 1.

2.2 Tree Representation of DTDs

A DTD is also represented as a labeled tree. In the tree representation, in order to represent optional elements, repeatable elements, and alternative of

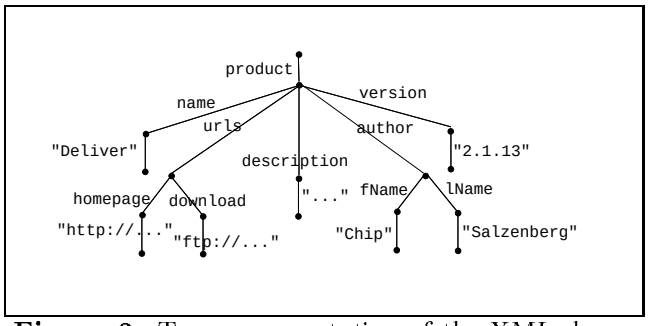

Figure 3: Tree representation of the XML document of Figure 1

elements, the set of operators $\mathcal{OP}$ = {`AND`, `OR`, ?, *} is introduced. The `AND` operator represents a sequence of elements, the `OR` operator represents an alternative of elements, the `?` operator represents an optional element, and the `*` operator represents repeatable elements (0 or more times). We remark that there is no need to introduce the `+` operator because an element type declaration like <!ELEMENT a (b)+> can be replaced by the equivalent one <!ELEMENT a (b,b*)>. Moreover, in the matching process we do not consider element declarations as <!ELEMENT a (b?)*>, <!ELEMENT a (b*)?>, <!ELEMENT a ((b*)*)?>, and so on, because all of them are equivalent to <!ELEMENT a (b)*>. Similarly, we do not consider <!ELEMENT a (((b)?)...)?> because it is equivalent to <!ELEMENT a (b)?>.

In our representation of DTDs each edge corresponds to an element tag, to an element type, or to an operator. In each tree that represents a DTD there is a single edge outgoing from the root, and the label of this edge belongs to $\mathcal{EN}$ (it is the name of the main element of documents described by the DTD). Moreover, there can be more than one edge outgoing from a node, only if the edge incoming to that node is labeled by `AND` or `OR`. Finally, all edges labeled by types are incoming to nodes that are leaves of the tree. Let $\mathcal{ET}$ be the set of possible basic types for elements ($\mathcal{ET}$ = {#`PCDATA`, `ANY`}). The following definition formally specifies the notion of DTD.

Definition 2 (*DTD*). A DTD T is a tree labeled on $\mathcal{EN} \cup \mathcal{ET} \cup \mathcal{OP}$ with the following properties:

1. $T = (v, [(l, T')])$ with $l \in \mathcal{EN}$;
2. for each $T' = (v, C)$ subtree of T, if l is the label of the edge incoming to T', then $l \in \mathcal{EN} \cup \mathcal{OP}$;
3. for each $T' = v'$ (leaf) subtree of T, if l is the label of the edge incoming to T', then $l \in \mathcal{ET}$;
4. for each $T' = (v, C)$ subtree of T, if $l \in$ {`OR`, `AND`} is the label of the edge incoming to T', then $C = [(l'_1, T'_1), \ldots, (l'_n, T'_n)], n > 1$;
5. for each $T' = (v, C)$ subtree of T, if $l \in \{?, *\} \cup \mathcal{EN}$ is the label of the edge incoming to T', then $C = [(l'_1, T'_1)]$. □

Figure 4 illustrates how the tree representation of a DTD is obtained. In particular, for each grammar rule that an XML DTD may follow, the figure shows the corresponding tree. Function *tree* is used to denote the tree corresponding to the element components. Figure 5 shows the tree representation of the `product` DTD of Figure 2.

[b]

We remark that the introduction of operators $\mathcal{OP}$ = {`AND`, `OR`, ?, *} allows us to represent the structure of all kinds of DTDs. The introduction of the `AND` operator is required in order

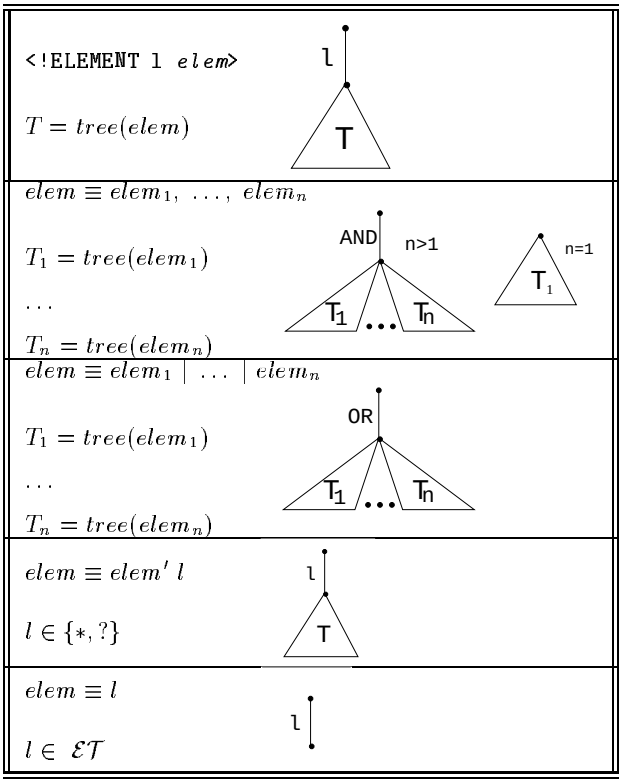

Figure 4: Tree representation of a DTD

to distinguish between an element containing an alternative between sequences (e.g. <!ELEMENT a(b|(c1,c2))>) and an element containing the alternative between all the elements in the sequence (e.g. <!ELEMENT a(b|c1|c2)>). The two different tree representations are shown in Figure 6(a,b). The document in Figure 6(c) is valid w.r.t. the DTD (b) but it is not valid w.r.t. the DTD (a).

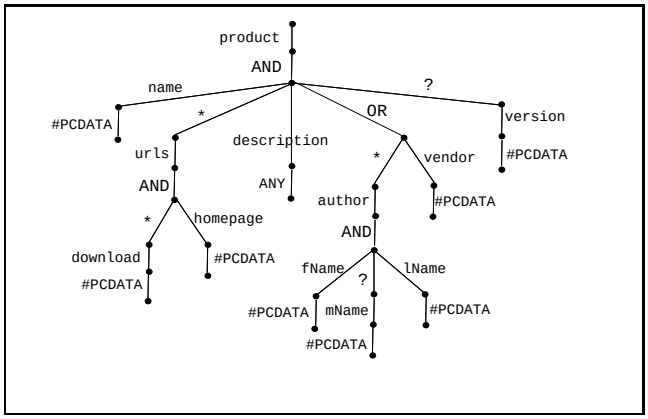

Figure 5: Tree representation of DTD of Figure 2

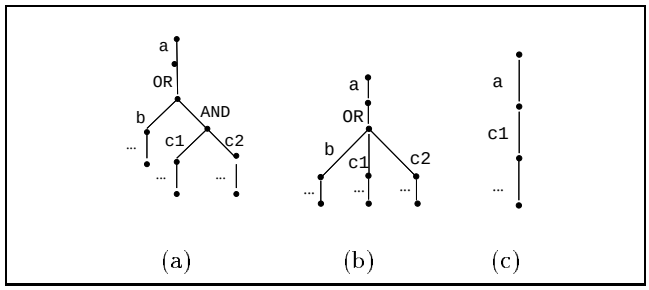

Figure 6: Example of DTDs motivating the introduction of AND labeled edges

Note finally that not only a document/DTD is a tree (as stated in the definitions), but any element of a document/DTD is a tree, subtree of the document/DTD. For instance, element name of the document in Figure 1 corresponds to the leftmost subtree rooted at the node reached by the product labeled edge in Figure 3.

2.3 Instance DTD

Because of the presence of alternative, repeteable and optional elements, a DTD actually describes a set of possible document structures. An instance DTD is a particular DTD describing a single document structure. This particular DTD only contains the AND operator, as formally stated by the following definition.

Definition 3 (*Instance DTD*). An instance DTD T is a DTD according to Definition 2 whose labels belong to $\mathcal{EN} \cup \mathcal{ET} \cup \{\texttt{AND}\}$. □

Documents conforming to an instance DTD exactly have the same structure. They can only differ for the values of data content elements. Given a DTD, a (possibly infinite) set of instance DTDs, that is, DTDs exactly describing the structure of each document valid for the DTD, can be associ-

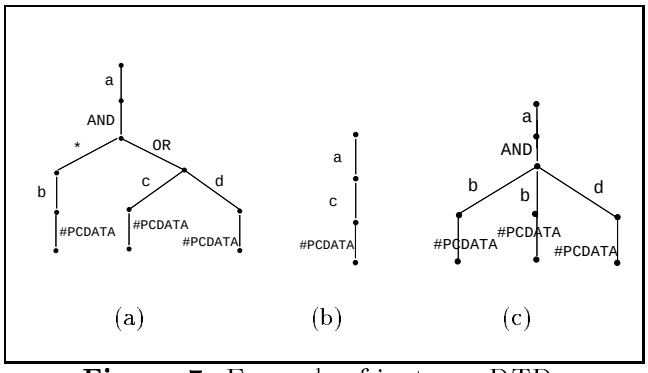

Figure 7: Example of instance DTDs

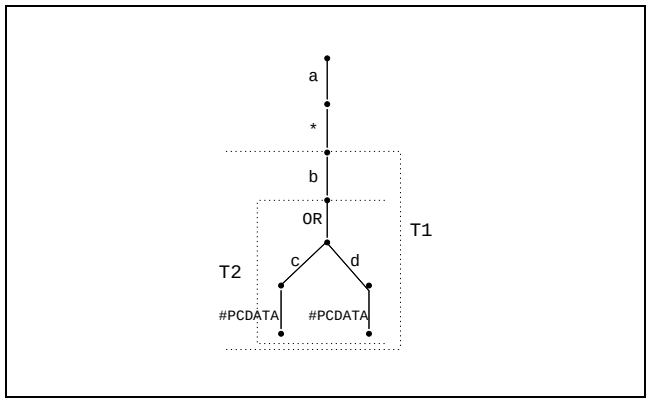

Figure 8: Repeatable element in a DTD

ated with it. The set of the instance DTDs associated with a DTD is infinite whenever the DTD contains a * operator.

Example 1 *DTDs in Figure 7(b) and 7(c) are examples of instance DTDs associated with the DTD in Figure 7(a).* ○

In a DTD tree some repeatable subtrees can be identified. Intuitively, a subtree is repeatable if, in the path from its root to the root of the tree, there is an edge labeled * before any edge labeled by an element name.

Definition 4 (*Repeatable Element*). Let T be a DTD, T_t a subtree of T, and $L = [l_1, \ldots, l_k]$ the list of labels on the edges of the path from the root of T_t to the root of T. T_t is repeatable if a label l_i exists, $1 \leq i \leq k$, such that $l_i = *$, and for all $l_j, j < i, l_j \notin \mathcal{EN}$. □

Example 2 *Consider the DTD whose tree representation is shown in Figure 8. Subtree* `T1` *is repeatable, because the first edge in the path from the root of* `T1` *to the root of the DTD is labeled by* *. *By contrast, subtree* `T2` *is not repeatable, because the first edge in the path from its root to the root of the DTD is labeled by* `b`*, that is an element name. This corresponds to the intuition that element* `b` *is repeatable an arbitrary number of times, whereas the content of* `b` *is not repeatable, that is, it can contain only a* `c` *element or a* `d` *element.* ○

Note that, according to the above definition, the tree corresponding to a terminal edge cannot be repeatable.

3 Requirements for a Similarity Measure

In this section we introduce through examples the requirements for a metric to measure the structural similarity between a document and a DTD.

3.1 Common, Plus, and Minus Elements

The similarity measure should consider: elements appearing both in the document and in the DTD, referred to as *common* elements; elements appearing in the document but not in the DTD, referred to as *plus* elements; elements appearing in the DTD but not in the document, referred to as

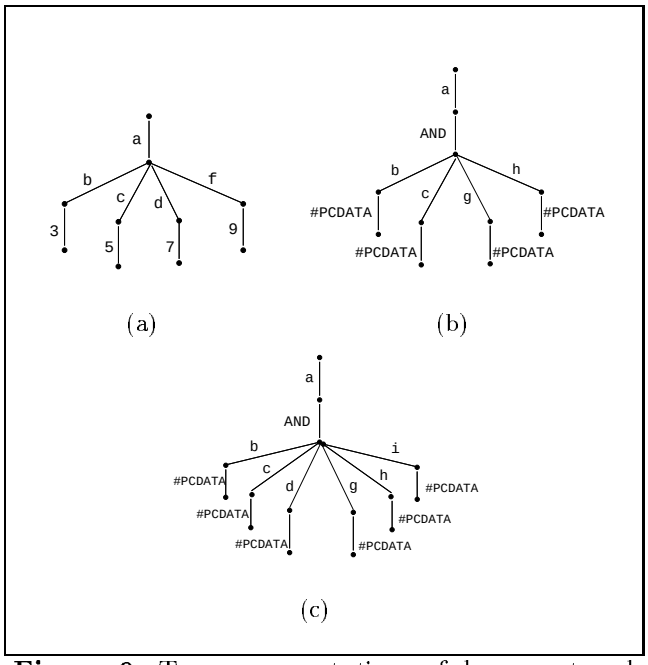

Figure 9: Tree representations of document and DTDs of Example 3

minus elements. Obviously, to achieve the best similarity, plus and minus elements should be minimized and common elements should be maximized.

Example 3 *Consider the document and DTDs whose tree representations are shown in Figure 9. The document in Figure 9(a) and the DTD in Figure 9(b) have the same tag for the document element but some of the subelements are different. In particular, the document and the DTD share elements* `b` *and* `c`*, whereas the document contains some elements,* `d` *and* `f`*, not appearing in the DTD, and the DTD contains some elements,* `g` *and* `h`*, not appearing in the document. Thus, there are two common elements, two minus elements, and two plus elements. Consider now the document in Figure 9(a) and the DTD in Figure 9(c). The document and the DTD share elements* `b`*,* `c`*, and* `d`*, whereas the document contains an element,* `f`*, not appearing in the DTD, and the DTD contains some elements,* `g`*,* `h`*, and* `i`*, not appearing in the document. Thus, there are three common elements, one plus element, and three minus elements. Evaluating the absence of an element as the presence of an additional element, the document is more similar to the DTD in Figure 9(c) because they have more common elements.* ○

3.2 Repeatable Elements

As we have discussed in Section 2 there can be many ways in which a document can be matched against a DTD. Because of alternative and repeatable elements, indeed, a DTD describes different possible document structures. When matching a document against a DTD, a structure, among those possible structures, must be identified, that is the most adequate for the document. In case of repeatable elements, the similarity measure must identify the best number of repetitions, that is, the one that maximizes common elements and minimizes plus and minus elements. Note that an higher number of repetitions can allow every element in the document to match with an element in the DTD (no plus) but can increase the number of unmatched elements in the DTD (minus).

Example 4 *Consider the document and DTD whose tree representations are shown in Figure 10. The possibility of repeating an arbitrary number of times the sequence of elements (*`b`*,* `c`*,* `d`*) allows us to map each element* `c` *in the document to a corresponding element in the DTD. However, since the document contains three* `c` *elements, the sequence*

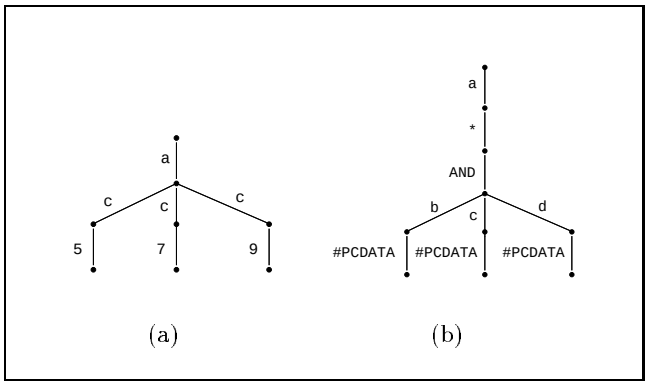

Figure 10: Tree representations of document and DTD of Example 4

in the DTD must be repeated three times, resulting in a total of nine elements: three present in the document (three `c` *elements) and six missing from the document (three* `d` *and three* `b` *elements). If, by contrast, we had repeated the sequence twice, we would have obtained two common elements and four minus elements. The situation is summarized in Table 1.* ○

repetitions	common	plus	minus
0	0	3	0
1	1	2	2
2	2	1	4
3	3	0	6
4	3	0	9

Table 1: Measuring the similarity between the document and the DTD of Example 4

3.3 Level of an Element

Till now we have considered all the elements in a document as "equivalent". However, elements at higher levels in the document structure are more relevant than subelements deeply nested in the document structure. The similarity measure should catch the intuition that the elements at a higher level in a document are more relevant than the elements at a lower level.

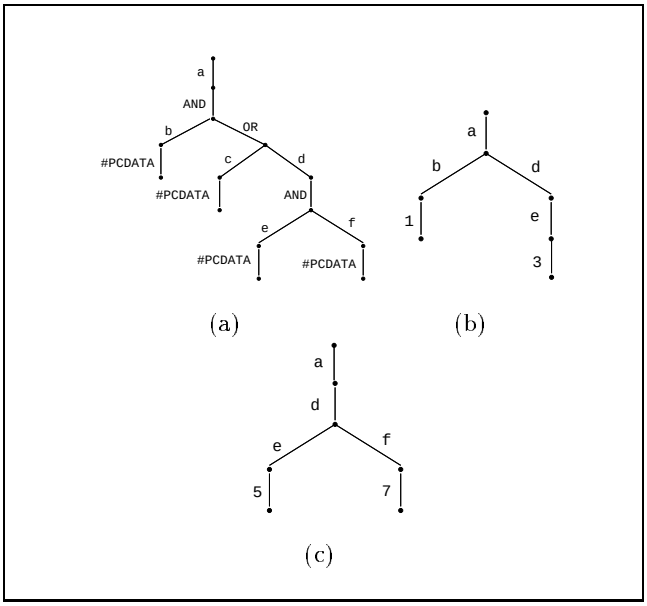

Figure 11: Tree representations of documents and DTD of Example 5

Example 5 *Consider the documents and the DTD whose tree representations are shown in Figure 11. Element* `f`*, subelement of element* `d`*, is missing in the document in Figure 11(b). By contrast, element* `b`*, subelement of element* `a`*, is missing from the document of Figure 11(c). The document in Figure 11(b) is more similar than the other to the DTD in Figure 11(a).* ○

We thus introduce the notion of level of an element, related to the depth of the corresponding tree. Given a tree T in a tree representing a document, the level of T is its depth as a tree, that is, the number of edges along the longest maximal path (that is, a path from the root to a leaf) in T. By contrast, given a tree T in a tree representing a DTD, its level is the number of edges, not labeled by an operator, along the longest maximal

path in T. This is because edges labeled by operators in DTD trees only influences the breadth of the corresponding document trees, not their depths. These notions are formalized by the following definition.

Definition 5 (*Function Level*). Let $T = (v, [(l_1, T_1), \ldots, (l_n, T_n)])$ be a subtree of a document or a DTD. Function *Level* is defined as follows:

$$Level(T) = max_{i=1}^{n} \mathcal{L}(l_i, T_i)$$

where:

$$\mathcal{L}(l, T') = \begin{cases} 0 & \text{if } T' = v \\ Level(T') & \text{if } l \in \mathcal{OP} \\ Level(T') + 1 & \text{otherwise} \end{cases}$$ □

Example 6 *Let T denote the tree of Figure 5, then $Level(T) = 3$.* ○

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

3.4 Weight of an Element

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

Example 7 *Consider the documents and DTDs whose tree representations are shown in Figure 12. If we match the document in Figure 12(a) against the DTD in Figure 12(d), the document lacks element* `c` *and the corresponding* `#PCDATA` *value. By contrast, if we match the document in Figure 12(a) against the DTD in Figure 12(b), the document lacks element* `c` *and the corresponding subtree. The lack of element* `c` *must be evaluated differently, since in the first case it has a simple data content, whereas in the second one it has a complex substructure. Consider now the document in Figure 12(c) and the DTD in Figure 12(d). The DTD specifies a* `#PCDATA` *content for element* `c`*, whereas in the document element* `c` *has a more complex substructure.* ○

The example above shows that the similarity measure should consider the structure of plus and minus elements. In case of minus elements, however, the structure is not fixed. Consider indeed element `c` of Figure 12(b): it presents an optional subelement (element tagged `d`) and an alternative of subelements (element tagged `f` or element tagged `g`). Our idea is to consider, as structure

of the minus elements, the simplest instance DTD associated with that portion of DTD. Thus, the measure should not take into account optional or repeatable elements and, in case of alternative, the measure should take into account only one of the alternative elements (reasonably the one with the simplest structure).

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

Definition 6 (*Function* $\mathcal{W}eight$). Let $T = (v, [(l_1, T_1), \ldots, (l_n, T_n)])$ be a subtree of a document or a DTD D, and w_l be the weight associated with a level in D. Function $\mathcal{W}eight$ is defined as follows:

$$\mathcal{W}eight(T, w_l) = \sum_{i=1}^{n} \omega((l_i, T_i), w_l)$$

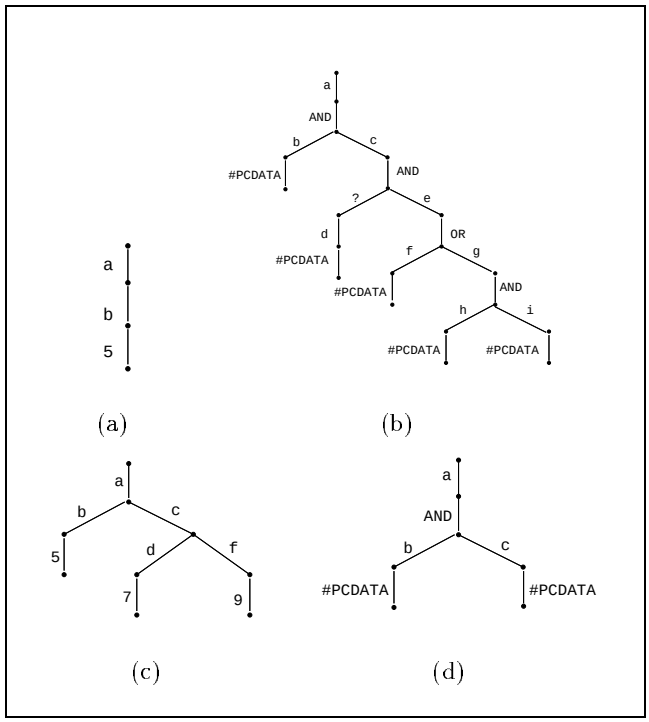

Figure 12: Tree representations of documents and DTDs of Example 7

where: $\omega((l, T'), w_l) =$

$$\begin{cases} w_l & \text{if } T' = v \\ 0 & \text{if } l \in \{*, ?\} \\ min_{i=1}^{n'} \omega((l'_i, T'_i), w_l) & \text{if } l = \texttt{OR} \text{ and } T' = \\ & (v', [(l'_1, T'_1), \ldots, (l'_{n'}, T'_{n'})]) \\ \mathcal{W}eight(T', w_l) & \text{if } l = \texttt{AND} \\ \mathcal{W}eight(T', \frac{w_l}{\gamma}) + w_l & \text{otherwise} \end{cases}$$ □

Example 8 *Let T denote the tree of Figure 5, recall that we consider $\gamma = 2$, $\mathcal{W}eight(T, 8) = 20$. Note that, in this example, the weight 8 is 2^3, where 3 is the number of levels of the DTD.* ○

4 Measuring Similarity

In the previous section we have outlined the requirements that a similarity measure should meet in order to properly evaluate the similarity between an XML document and a DTD. In

this section we present a similarity measure addressing such requirements. We first describe the evaluation function we employ, then present the matching algorithm, and finally define the similarity measure.

4.1 Evaluation Function

Intuitively, in order to evaluate the similarity between a document D and a DTD T we need to consider the set of instance DTDs associated with T and select the "best one", that is, the one that maximize an *evaluation function*. Once selected the most similar instance DTD, the similarity between it and the document can be measured as follows.

1. Moving down from the top level, for each level l of document D a triple (p, m, c) is determined representing the number of plus, minus, and common elements at level l w.r.t. the "best" instance DTD. Note that, if a plus element is found at a level l, all its subelements are plus (analogously for minus elements).

2. Each triple determined at step 1 is multiplied for the level weight.

3. A single triple is obtained by summing together the triples obtained at step 2.

A degree of similarity is then computed by means of the following function, based on the triple obtained at step 3.

Definition 7 (*Function* $\mathcal{E}$). Let (p, m, c) be a triple of natural numbers and α, β be real numbers s.t. $\alpha, \beta \geq 0$. Function $\mathcal{E}$ is defined as follows:

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

Function $\mathcal{E}$, defined according to the *ratio model* [15], is based on two parameters α, β and returns a value between 0 and 1. Depending on the value assigned to these parameters, the function gives more relevance to *plus* elements w.r.t. *minus* elements, or vice-versa. For example, if $\alpha = 0$ and $\beta = 1$ the function does not take into account *plus* elements in measuring similarity. Therefore, a document with only extra elements w.r.t. the ones specified in the DTD has a similarity measure equal to 1. By contrast, if $\alpha = 1$ and $\beta = 0$ the evaluation function does not take into account *minus* elements in the similarity measure. In the following examples we assume that $\alpha = \beta = 1$, thus giving the same relevance to *plus* and *minus* elements. Moreover, as we will discuss further, if $\alpha = \beta = 1$ the fact that the result of the evaluation of a triple (p, m, c) is 1 is related to the validity of a document w.r.t. a DTD.

Example 9 *Consider the document D and the DTD T whose tree representations are shown in Figure 13(a) and 13(b), respectively. Figure 13(c) shows the best instance DTD for D among the ones that can be generated starting from the DTD T.*[1] *Moreover, for each level, the level weight is shown together with a triple (p, m, c) representing the number of plus, minus, and common elements multiplied for the level weight. By apply-*

[1] For the sake of simplicity we omit the `AND` labeled edges.

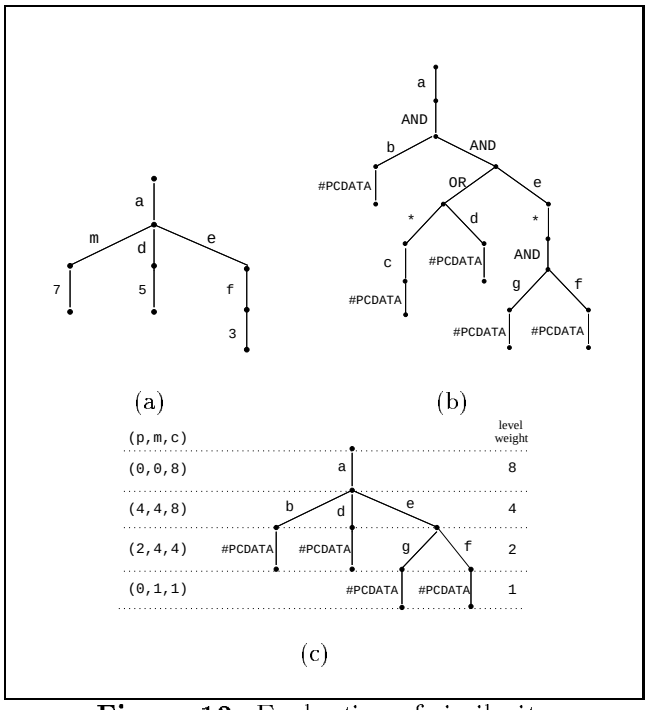

Figure 13: Evaluation of similarity

ing function $\mathcal{E}$ to the sum of the triples associated with each level ($(6, 9, 21)$) we obtain the value $0,58$ *which represents the highest similarity degree between the document D and the DTD T.* ○

As we have outlined in Section 2 the set of instance DTDs associated with a DTD may be infinite, when the DTD has repeatable elements. However, if we take into account the document against which the DTD should be matched, we can restrict to a finite[2] set of instance DTDs, that we refer to as instance DTDs associated with a DTD given a document. This follows from the intuition that a repeatable subtree of a DTD can be repeated until there are elements in the document that are matched. Further repeating the subtree is useless.

[2]Note that, however, the set can be huge when the DTD has a complex structure.

4.2 Matching Algorithm

The naive solution of evaluating the similarities between the document and each of the instance DTDs associated with the DTD given the document, to select the highest, is however expensive. Thus, an algorithm determining the best similarity degree without computing the set of instance DTDs has been developed. Such algorithm is based on the idea of locally determining the best structure for a DTD element, for elements containing alternatives or repetitions, as soon as the information on the structure of its subelements in the document is known.

The algorithm is general enough to handle all possible cases. However, for the sake of clarity, in the remainder of the section a simplified version of the algorithm will be presented. This simplified version is based on the assumption that, in the declaration of an element, two subelements with the same tag are forbidden. That is, element declarations such as <!ELEMENT a (b*, (c|b))> are not handled. In the following section we will show how this assumption can be removed.

The algorithm, implemented through a recursive function $\mathcal{M}$, associates a triple (p, m, c) with a document D and a DTD T. Function $\mathcal{M}$ is shown in Appendix A (Figure 17). The inputs of $\mathcal{M}$ are a document D, a DTD T, a level weight w_l, and a flag rp indicating the repeatability of T. The level weight w_l initially is the maximum between the number of levels of the document and those of the DTD, and is then divided by γ each time the function moves down a level in D. This ensures

that the similarity evaluation properly assigns a level weight to each level of the two structures.

The idea behind the behavior of function $\mathcal{M}$ is to visit the document and the DTD, at the same time, from the root to the leaves, to match common elements. Specifically, two distinct phases can be distinguished:

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

In what follows we describe the main ideas behind the two phases, and discuss base cases for termination.

First phase. Starting from a pair (T_d, T'_t), with T'_t of the form $(v, [(l, T_t)])$, the behavior of function $\mathcal{M}$ (and the sequence of recursive calls) depends on l. If $l = ?$, $\mathcal{M}$ is called on the pair $(T_d,\ T_t)$ as if the ? were not present. If $l = *$, $\mathcal{M}$ is called on the pair $(T_d,\ T_t)$ marking T_t as repeatable. If $l =$ `AND` or $l =$ `OR`, and T_t is of the form $(v, [(l_1, T_1), \ldots (l_n, T_n)])$, $\mathcal{M}$ is called on the pairs $(T_d, (v, [(l_i, T_i)]))$, $1 \leq i \leq n$. If $l \in \mathcal{EN}$ is the name of an element, the subtrees of T_d reached through an edge labeled l are computed. Let $\{T_1^{d'}, \ldots, T_k^{d'}\}$ be the set of these subtrees. If $k > 0$, i.e. there are common elements between the two structures, $\mathcal{M}$ is called on the pairs $(T_i^{d'}, T_t)$, $1 \leq i \leq k$. Intuitively, we move down a level in the document only for those elements that appear in the DTD. Through the recursive calls of function $\mathcal{M}$ it is thus possible to identify the common elements between the two structures and the parts of the DTD, containing elements that are present in the document, that are repeatable. Note that no evaluation of plus, minus, and common elements is performed in this phase.

Base cases. A triple (p, m, c) is computed when a terminal edge of the DTD is reached or when a subtree of an element in the DTD that does not match with any element in the document is found. This triple represents the evaluation of plus, minus, and common elements for base cases. For instance, if a terminal edge labeled by `#PCDATA` is matched against a terminal edge of the document, the triple $(0, 0, w_l)$ is returned, corresponding to the fact that there are no plus and no minus elements, whereas there is a common (terminal) element whose weight is the level weight.

Second phase. Starting from the evaluation of base cases, on return from the recursive calls of $\mathcal{M}$, the plus, minus and common elements of an intermediate pair (T_d, T_t) can be evaluated, taking into account the label l of the edge outgoing from T_t and its repeatability. At the end, a triple (p, m, c) is associated with the pair (D, T).

Actually, computing a single triple (p, m, c) is not enough to evaluate all the intermediate pairs. In some cases a list of (p, m, c) triples must be computed. These lists are generated when a subtree

is repeatable and the decision of the best number of repetitions cannot be taken at that level of the tree, rather it needs to be postponed at an higher level. The cardinality of the list associated with the pair (T_d, T'_t) depends on the repeatability of T'_t and on the label l of the edge outgoing from the root of T'_t. Specifically, the cardinality of the list is one when T'_t is not repeatable, or when T'_t is repeatable and $l \in \mathcal{V} \cup \mathcal{ET} \cup \{*\}$. By contrast, it can be greater than one when T'_t is repeatable and $l \in \mathcal{EN} \cup \{\texttt{AND}, \texttt{OR}, ?\}$.

Due to space limitations we cannot discuss how the (p, m, c) triples are generated in all the cases (that depend on the label of the subtree of the DTD). However, we focus on one of the most interesting cases: a pair $(T_d, (v, [(l, T_t)])$, with l the label of an element. As outlined above the set $\{T_1^{d'}, \ldots, T_k^{d'}\}$ of children of T_d labeled by l is computed.

If this set is empty, i.e. the element of the DTD tagged l is missing in the document, a (p, m, c) triple is determined depending on the repeatability of T_t. If T_t is not repeatable, this means that the subtree were required in a valid document, but it is missing. Therefore, function $\mathcal{W}eight$ is called on T_t in order to determine the weight m of the missing element and its subtree. The triple $(0, m, 0)$ is returned. By contrast, if T_t is repeatable no real evaluation can be done at this stage of the computation, thus a dummy triple $(0, 0, 0)$ is returned. If this element is actually missing in the document, this will be detected later (when evaluating the outer element).

If, by contrast, the set $\{T_1^{d'}, \ldots, T_k^{d'}\}$ of elements tagged l in T_d is not empty, let the recursive call of $\mathcal{M}$ on each of the children $T_i^{d'}$ and T_t return the triple (p_i, m_i, c_i), representing the evaluation of common, minus, and plus elements found in the subtrees. Two further cases can be distinguished, depending on the repeatability of T_t. If T_t is not repeatable, only one of the $T_i^{d'}$ is allowed in the document, and the others are exceeding. Thus, first of all the triple (p, m, c), representing the best matching, that is, the triple maximizing function $\mathcal{E}$, is selected. The triple to be returned is then determined as follows:

- m represents the evaluation of the minus elements of the selected child, thus it is not modified;
- c represents the evaluation of the common elements of the selected child, thus the level weight w_l is added to it, since at the current level a common element has been found;
- p represents the evaluation of the plus elements of the selected child, thus we add to it:
  - the plus and common components of the triples that we have not selected, plus the level weight, since each child that we have not selected must be evaluated as a plus, and
  - the weights of the subtrees of each $T_i^{d'}$ that had no matching in T_t (those labeled by a label not appearing at that level in the DTD), because they had not been considered yet, but they represent plus components.

By contrast, if T_t is repeatable, no decision can be taken at this level of the computation, thus the

level weight is addeded to the c component of each of the triple and the list of triples obtained are returned, ordered on the base of function $\mathcal{E}$. The optimal number of occurrences of this element will be established later (when evaluating the outer element).

4.3 Similarity Measure

Based on function $\mathcal{M}$, the similarity measure between a document and a DTD is defined as follows.

Definition 8 (*Similarity Measure*). Let D be a document and T a DTD. Let d be the maximum between the levels of D and T ($d = max(Level(D), Level(T))$. The similarity measure between D and T is defined as follows:

$$\mathcal{S}(D,T) = \mathcal{E}(\mathcal{M}\langle D, T, \gamma^d, \mathtt{F}\rangle) \qquad \square$$

Example 10 *Figure 14 shows how function $\mathcal{M}$ works on the document and DTD shown in Example 9. An edge (v, v') of the tree is bold if a recursive call of function $\mathcal{M}$ has been made on the subtree rooted at v'. If an edge is bold the label is followed by the triple (p, m, c) obtained from the evaluation of the corresponding subtree. If an edge is not bold, but the label is followed by a triple (p, m, c), it represents the evaluation of minus elements of the subtree.*[3] *The triple (p, m, c) associated with the document and the DTD is exactly the same obtained with the naive algorithm.* $\bigcirc$

[3] Note that, in this case the triple is $(0, 0, 0)$ if the corresponding subtree is repeatable.

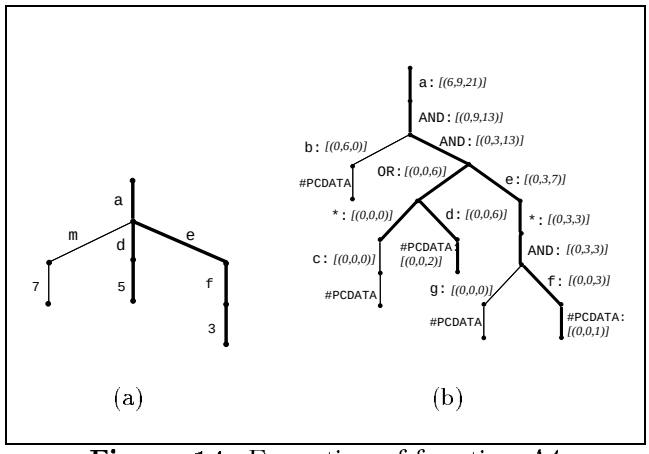

Figure 14: Execution of function $\mathcal{M}$

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

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

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

5 Discussion

In this section we discuss how the technique can handle arbitrary XML documents, and the experimental results obtained.

5.1 Extension to Arbitrary XML Documents

Though we have presented an algorithm that works for a subset of XML documents, the approach can handle arbitrary XML documents. In what follows we discuss the various issues not covered till now: attributes, elements with the

same tag at the same level in the DTD, and tag similarity rather than tag equality.

Attributes. Attribute handling is very simple: DTDs and documents are represented as trees with two different kinds of edges: attribute and element edges (with appropriate labels), and the algorithm handles attributes as it handles elements (attributes can be seen as data content elements, i.e., they cannot be nested). Empty elements are handled by introducing, in the tree representation, a special `"EMPTY"` value used to label the edge corresponding to the data content of an empty element. Then, they are handled exactly as simple elements with a `#PCDATA` content. When considering attributes, the issue arises of whether or not to allow an attribute to match with a subelement containing only text. The choice of using attributes rather than elements to model a short string of text mainly depends on the developer skills and experience. Since in some other situations, however, the distinction between elements and attributes could be relevant, in our approach, one can specify whether the matching of an attribute with a data content element can be considered. Together with α, β, γ this is another parameter of our measure.

Subelements with the same tags. As discussed in Section 4 the algorithm we have presented is based on the assumption that in the declaration of an element in the DTD two different subelements with the same tag cannot appear. This assumption simplifies the algorithm, since all the occurrences of a certain element in the document can match with at most one element in the DTD. By contrast, when the previous assumption does not hold, different matches are possible for an element in the document. All these possible matches must be considered and evaluated, and the best one (that is, the one leading to the highest similarity) must be selected.

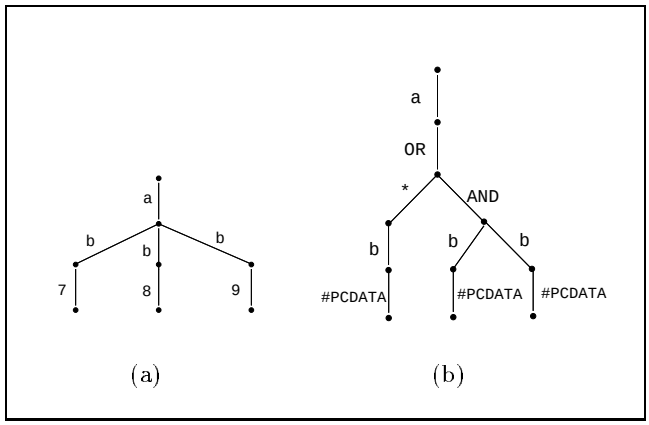

Figure 15: Tree representations of document and DTD of Example 11

Example 11 *Consider the document and DTD of Figure 15. Different matches are possible for the* `b` *elements in the document. Some of these matches (e.g. the one that matches the three* `b`*'s with the* `*` *subtree) lead to a perfect matching (no plus and no minus) and are thus better than the others (e.g. the one that matches the three* `b`*'s with the* `AND` *subtree).* ○

In the algorithm this is handled by considering and evaluating all these possible matching. A matrix is built for each element in the DTD with some direct subelements with the same tag. This matrix has a column for each of the subelements of an element with the same tag, and it contains one row for each of the possible ways in which the elements with that tag at that level in the document

can be matched with them. The similarity measure is evaluated for all these possible matching, and the best one is then selected. Note that it is not possible to choose the best matching without computing the similarity between the subtrees of the elements in the document and in the DTD, since in the documents there is no constraint on the element content (i.e. elements with the same tag can have contents with different structures). However, since a single definition for element `b` can be given in the DTD, the subtrees under elements `b` in the DTD are the same.[4] Thus, each subtree under element `b` in the document can be matched against only one of them. The (p, m, c) triple obtained represents its similarity with the definition of element `b` in the DTD. This similarity is evaluated only once. Thus, the subtree does not need to be visited again.

Tag similarity. Though we have considered tag equality in discussing our technique, tag similarity can be considered as well. Since we are weakening the notion of conformance for what concerns document structure, the requirement of tag equality can be weakened as well. Tag similarity can be evaluated relying on a Thesaurus [11]. The Thesaurus we refer to stores two main terminological relationships:[5]

- USE: it is defined between terms that are considered equivalent, because they are morphological variants (*stems*). For example: an acronym term and its expansions, two differently spelled versions of a term, a singular term and its plural.
- SYN (synonymous terms): it is defined between terms considered as synonyms. In most cases, these terms denote the same category of real world objects (e.g., "Client" and "Customer"). A term and its translations in different languages also belong to this relationship.[6]

Different affinity values δ can be assigned to each terminological relationship. Let δ_{SYN} and δ_{USE} denote these values. Then $0 \leq \delta_{SYN} \leq \delta_{USE} \leq 1$. Typical values we refer to are $\delta_{USE} = 1$ and $\delta_{SYN} = 0.8$ (as the ones considered in [7]). In the similarity measure, when matching two element tags, the value $1 - \delta$ is added to the component m of the subtree evaluation. In this way we capture the missing tag equality.

Tag similarity can be handled by extending the approach taken for DTDs with different elements with the same tag at the same level. Also in this case, indeed, different matches are possible for an element with that tag in the document. All the possible matches must be considered and evaluated, and the best one (that is, the one leading to the highest similarity) must be selected. In this case, however, the DTD could contain elements,

[4] Actually, in this case, a DAG is a better representation for the DTD. For the sake of simplicity we refer to the tree representation introduced in Section 2.

[5] Broadening/narrowing of terms, and the inclusion of related terms can be considered as well.

[6] Alternatively, a separate relationship TR can be considered for translations, without affecting the approach.

subelements of the same element, whose tags are synonyms, but whose structures (and thus subtrees) are different. Thus, each subtree under an element tagged with a synonym of that tag in the document must be matched against all of them, resulting in the same subtree in the document being visited several times.

Example 12 *Consider the document and DTD of Figure 16. Suppose* $\texttt{b}_1$, $\texttt{b}_2$, $\texttt{b}_3$ *be synonyms. Different matches are thus possible for the* `b` *elements in the document. One of these matches (the one that matches* $\texttt{b}_1$ *and* $\texttt{b}_2$ *with the* `AND` *subtree and* $\texttt{b}_3$ *with the* `*` *subtree) leads to a perfect matching (no plus and no minus) and is thus better than the others (e.g. the one that matches all the three* `b` *with the* `AND` *subtree).* ○

The complexity of the algorithm we have presented in Section 4/Appendix A is polinomial in the number of nodes of the document and of the DTD. Each node in the document and each node in the DTD is indeed visited only once, and for each node in the DTD, in the worst case, some operations that are quadratic in the number of nodes of the document are performed. In the general case discussed above, however, the algorithm is exponential. Thus, there is an obvious trade-off between the efficiency of the matching process and the strictness of the similarity requirements.

5.2 Experimental Results

The described algorithm as well as its extensions discussed above have been implemented in Java,

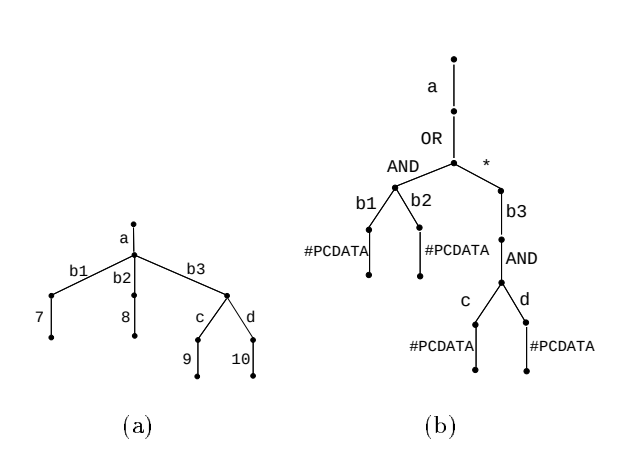

Figure 16: Tree representations of document and DTD of Example 12

using the DOM libraries [16]. To validate the proposed technique we have performed some experiments over "real data", gathered from the Web, and "synthetic data", randomly generated. Because of space limitations, in what follows, we only briefly report some of these experiments. A more detailed description of the experiments and their results can be found in [4].

As remarked in [6] we are still lacking large XML repositories: the generation of sources of XML data for evaluation purposes as well as the development of evaluation methods is indeed mentioned as the first research issue to address. Thus, also in the case of "real data" we started from HTML documents and we have then extracted XML documents from them. Data come from two sources of related documents on the Web. These sources[7] contain HTML documents describing software products and their features. By means of structure extraction tools (like XWrap [10], Nodose [1]) we have obtained about 400 XML

[7] `www.knolwledgecenters.com`, `www.webcompare.com`

documents from each source. We have then randomly built 5 groups of 5 XML documents each, and for each group we have generated the DTD describing the structure of documents in the group, thus obtaining 5 DTDs. The document and DTD of Figures 1 and 2 are excerpts of documents extracted from those sources and of DTDs developed with the method outlined above.

Synthetic data are generated by the means of a random generator we have developed. We generated 10000 documents that contain elements arbitrarily choosen from the Dublin Core element set [9]. Each element, representing information about a resource (e.g. `title`, `creator`, `subject`), is optional, repeatable, and may appear in any order. Moreover, they can have qualifiers, i.e. subelements, refining their meaning. For example, the `creator` element can have the `personalName` subelement specifying the name of the creator. We have then specified 3 DTDs using subsets of the Dublin Core element set.

In both the experiments, matching all the documents in the sources against the DTDs, we have obtained the following results:

- each time a document is valid for a DTD, the matching algorithm correctly returns 1 as similarity;
- for each document D, and for each pair of DTDs T_1, T_2 such that D is not valid neither for T_1 nor for T_2, whenever $\mathcal{S}(D, T_1) > \mathcal{S}(D, T_2)$, D actually is more similar to T_1 than to T_2.

From the experiments also emerged that, if two DTDs T_1 and T_2 obtain close similarity values with each document (that is, for each D $\mathcal{S}(D, T_1) \simeq \mathcal{S}(D, T_2)$) they likely correspond to the same class of documents, thus they actually belong to a single structural cluster.

The previous experiments have been carried on by fixing $\alpha = \beta = 1$, $\gamma = 2$. Some additional experiments have been carried on fixing $\alpha = 0$. In this case, the matching process can be seen as a method to evaluate the structural query that allows one to retrieve the documents that contain the elements specified in the DTD. The documents with similarity degree equal to 1 are those containing all the elements required by the DTD, and potentially additional elements. The similarity measure has also been used to rank documents based on their structural similarity with the DTD, applying the idea of ranking queries [2] to document structure.

More experiments are currently under development to compare the results with those obtained from the technique with different values of the parameters, and with alternative definitions of function $\mathcal{E}$.

6 Conclusions

In the paper we have proposed a measure to evaluate the structural similarity of a document w.r.t. a DTD. Such measure can be used for classifying a document in a set of DTDs, choosing the DTD with which the document has the highest degree of similarity. Moreover, this measure can be used to structurally clustering XML documents, resulting in more effective and efficient searches on the documents. Our approach complements the exist-

ing techniques to obtain from a set of XML documents a (minimal) set of DTDs describing their structures [8, 13].

The work described in this paper can be extended in different directions. First, we are investigating how to use the feedback information of the matching algorithm to obtain a set of DTDs describing in the most accurate way the structure of the considered XML documents. From the clustering viewpoint, this corresponds to look for clusters with maximal intra-cluster similarity and minimal inter-cluster similarity. From the classification viewpoint, this can be seen a sort of *schema evolution* process by which the structural information in a source is rationalized.

Another direction we are investigating is the integration of structure-based retrieval with classical content-based retrieval. The goal is to express and evaluate queries containing both filters on the document structure and on the document content. In this context, we also plan to investigate how the structural clustering can be integrated with traditional, content-based, clustering techniques.

References

[1] B. Adelberg and M. Denny. Nodose 2.0. In *Proc. ACM Int'l Conf. on Management of Data*, pages 559–561, 1999.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] E. Bertino, G. Guerrini, I. Merlo, and M. Mesiti. An Approach to Classify Semi-Structured Objects. In *Proc. European Conf. on Object-Oriented Programming*, LNCS 1628, pages 416–440, 1999.

[4] E. Bertino, G. Guerrini, M. Mesiti, I. Rivara, and C. Tavella. Measuring the Structural Similarity among XML Documents and DTDs, 2001. http://www.disi.unige.it/person/MesitiM.

[5] R. Bourret. XML and Databases, 2001. http://www.rpbourret.com/xml/.

[6] D. Carmel, Y. Maarek, and A. Soffer. XML and Information Retrieval: a SIGIR 2000 Workshop. *SIGMOD Record*, 30(1):62–65, March 2001.

[7] S. Castano, V. De Antonellis, M. G. Fugini, and B. Pernici. Conceptual Schema Analysis: Techniques and Applications. *ACM Transactions on Database Systems*, 23(3):286–333, Sept. 1998.

[8] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proc. ACM Int'l Conf. on Management of Data*, volume 29, pages 165–176. 2000.

[9] Dublin Core Initiative. http://dublincore.org/.

[10] L. Liu, C. Pu, and W. Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Int'l Conference on Data Engeneering*, pages 611–621, 2000.

[11] A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, Nov. 1995.

[12] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proc. of Int'l Conf. on Very Large Data Bases*, pages 122–133, 1998.

[13] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In

Proc. of ACM Int'l Conf. on Management of Data, pages 295–306, 1998.

[14] G. Salton, C. Yang, and C. Yu. A Theory of Term Importance in Automic Text Analysis. *Journal of the Americal Society for Information Sciences*, 26(1):33–44, 1975.

[15] A. Tversky. Features of Similarity. *Journal of Psychological Review*, 84(4):327–352, 1977.

[16] W3C. Document Object Model (DOM), 1998.

[17] W3C. Extensible Markup Language 1.0, 1998.

A Matching Algorithm

Table 2 shows the notation used in the algorithm to handle lists of triple (p, m, c). Moreover, given a tree in a document or DTD, function $\mathcal{K}ids$ returns the set of its element children. Then, function $\mathcal{K}ids_{|l}(T)$, returns the subset of children of T that are reached through an edge labeled l. Given a tree T in a document or DTD, function $\alpha\beta$ returns the list of tags associated with the subelements of T. Note that, whenever function $\alpha\beta$ is applied to a node in a DTD, it returns the subelements independently from the operators used in the element type declaration. Moreover, function $\alpha\beta$ returns an empty set when applied to a tree that does not have edges labeled by an element name. For instance, if T denotes the tree of Figure 5, then $\alpha\beta(T)$ = [`name`, `urls`, `description`, `author`, `vendor`, `version`].

Figure 17 shows the algorithm $\mathcal{M}$ computing the similarity between a document and a DTD. Figure 18 and Figure 19 show two auxiliary functions used that the algorithm $\mathcal{M}$. Function $\mathcal{H}_{\texttt{AND}}$ is used to combine the results of evaluation the subtrees of a tree T_t, labeled `AND`, when T_t is repeatable. It helps to identify the possible repetition of the `AND` operator. Function $\mathcal{H}_{\texttt{OR}}$ is analogous to the previous function for a tree labeled by the `OR` operator.

function	description $(r = [(p_1, m_1, c_1), \ldots (p_n, m_n, c_n)])$
$\#\{r\}$	cardinality of r $(\#\{r\} = n)$
$r[i]$	(p, m, c) of position i in r
$Sort_F(r)$	r ordered on the base of function F
$max_F(r)$	$(p, m, c) \in r$ s.t. $F((p, m, c))$ is max
$last(r)$	last (p, m, c) in r
$append(r, (\bar{p}, \bar{m}, \bar{c}))$	$[(p_1, m_1, c_1), \ldots (p_n, m_n, c_n), (\bar{p}, \bar{m}, \bar{c})]$
$deleteHead(r)$	r without the first (p, m, c)
$Normalize(r) = r'$	$r'[1] = r[1]$ and for $i = \#\{r\}$ downto 2, $r'[i] = (p_i, m_i - m_{i-1}, c_i - c_{i-1})$

Table 2: Notations used in the $\mathcal{M}$ Algorithm

Algorithm $\mathcal{M}(T_d : \mathcal{DOC}, T'_t : \mathcal{DTD}, rp : Bool, w_l : \mathbb{N}) : list\_of(\mathcal{PMC})$
begin /* T'_t *is of the form* $(v_t, [(l, T_t)])$ */
  case of (l)
  `#PCDATA`: **if** (T_d corresponds to a terminal edge) **return** $[(0, 0, w_l)]$ **else return** $[(\mathcal{W}eight(T_d, w_l), w_l, 0)]$;
  `ANY`: **return** $[(0, 0, \mathcal{W}eight(T_d, w_l))]$;
  $l \in \mathcal{EN}$: $\{T_1^{d'} \ldots T_k^{d'}\} = \mathcal{K}ids_{|l}(T_d)$
    if $(k > 0)$ **begin**
      for $i = 1$ **to** k **do** $[(p_i, m_i, c_i)] = \mathcal{M}\langle T_i^{d'}, T_t, \frac{w_l}{\gamma}, rp\rangle$
      if $(rp =$ `T`$)$ **return** $Sort_{\mathcal{E}}([(p_1, m_1, c_1 + w_l), \ldots, (p_k, m_k, c_k + w_l)])$;
      else begin
        $(p, m, c) = max_{\mathcal{E}}[(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)]$;
        $\bar{p} = \sum_{i+1}^{k}(p_i + c_i) + k \cdot w_l - (c + w_l)$
        $A = \bigcup_{i=1}^{k}\{T | T \in \mathcal{K}ids_{|\bar{l}}(T_i^{d'}), \bar{l} \in \alpha\beta(T_i^{d'}) \setminus \alpha\beta(T_t)\}$;
        $\tilde{p} = \sum_{T \in B}(\mathcal{W}eight(T, \frac{w_l}{\gamma}) + w_l)$
        return $[(\bar{p} + \tilde{p}, m, c + w_l)]$;
      end
    end
    else if $(rp =$ `T`$)$ **return** $[(0, 0, 0)]$; **else return** $[(0, \mathcal{W}eight(T_t, \frac{w_l}{\gamma}) + w_l, 0)]$;
  `*`: $[(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)] = \mathcal{M}\langle T_d, T_t, w_l,$ `T`$\rangle$;
    /* T_t *is of the form* $(v'_t, [(l', \bar{T}_t)])$ */
    if $(l' \in \mathcal{EN})$ **return** $[\sum_{i=1}^{k}(p_i, m_i, c_i)]$; **else return** $[max_{\mathcal{E}}\{(p_1, m_1, c_1), \ldots, (p_k, m_k, c_k)\}]$;
  `?`: $r = \mathcal{M}\langle T_d, T_t, w_l, rp\rangle$;
    if $(rp =$ `T`$)$ **return** r;
    else /* r *is of the form* $[(p, m, c)]$ */
      if $(c = 0)$ **return** $[(p, 0, 0)]$; **else return** $[(p, m, c)]$;
  `AND`: `OR`: /* T_t *is of the form* $(v'_t, [(l_1, T_1^t), \ldots, (l_n, T_n^t)])$ */
    for $i = 1$ **to** n **do** $r_i = \mathcal{M}\langle T_d, (v'_t, [(l_i, T_i^t)]), w_l,$ `T`$\rangle$;
    if $(rp =$ `T`$)$ **begin**
      if $(l =$ `AND`$)$ $r = \mathcal{H}_{\texttt{AND}}([\langle l_1, T_1, r_1\rangle, \ldots, \langle l_n, T_n, r_n\rangle], w_l)$; **else** $r = \mathcal{H}_{\texttt{OR}}([\langle l_1, r_1\rangle, \ldots, \langle l_n, r_n\rangle])$;
      return r;
    end
    else /* r_i *of the form* $[(p_i, m_i, c_i)]$ */
      if $(l =$ `AND`$)$ **return** $[\sum_{i=1}^{n}(p_i, m_i, c_i)]$;
      else begin
        $(p, m, c) = max_{\mathcal{E}}[(p_1, m_1, c_1), \ldots, (p_n, m_n, c_n)]$;
        $\bar{p} = \sum_{i=1}^{n}(p_i + c_i) - c$;
        return $[(\bar{p}, m, c)]$;
      end
  end-case
end

Figure 17: The $\mathcal{M}$ algorithm

Function $\mathcal{H}_{\texttt{AND}}(r : list\_of(l : \mathcal{EN} \cup \mathcal{OP}, T : tree, \bar{r} : list\_of(\mathcal{PMC})), w_l : \mathbb{N}) : list\_of(\mathcal{PMC})$
begin
  Let $r = [\langle (l_1, T_1), r_1 \rangle, \ldots, \langle (l_n, T_n), r_n \rangle]$; **Let** $A = \{\#\{r_i\} \mid 1 \leq i \leq n, r_i[1] = (\bar{p}, \bar{m}, \bar{c}), \bar{c} \neq 0\}$;
  if $(A = \emptyset)$ **return** $[(0,0,0)]$; **else** $rp_{max} = max(A)$;
  $list = []$;
  for $i = 1$ **to** rp_{max} **do**
  begin
    $(p, m, c) = (0, 0, 0)$;
    for $j = 1$ **to** n **do**
      begin
        Let $r_j = [(p_1^j, m_1^j, c_1^j), \ldots, (p_k^j, m_k^j, c_k^j)]$ and $k = \#(r_j)$;
        If $(c_1^j = 0$ **and** $l_j \notin \{*, ?\})$ $(p, m, c) = i * \mathcal{W}eight(T_i, w_l)$;
        else **case of** (l_j)
          $(l_j \in \mathcal{OP} \setminus \{*, ?\})$:
            if $(k \geq i)$ $(p, m, c) = (p, m, c) + (p_i^j, m_i^j, c_i^j)$;
            else $(p, m, c) = (p, m + (i - k) * \mathcal{W}eight(T_j, w_l), c) + last(r_j)$;
          $(l_j = *)$:
            $(p, m, c) = (p, m, c) + (p_1^j, m_1^j, c_1^j)$;
          $(l_j = ?)$:
            if $(label(T_j) \in \mathcal{OP})$
              if $(k \geq i)$ $(p, m, c) = (p, m, c) + (p_i^j, m_i^j, c_i^j)$;
              else $(p, m, c) = (p, m, c) + last(r_j)$;
            else
              if $(k \geq i)$ $(p, m, c) = (p, m, c) + \sum_{h=1}^{i}(p_h^j, m_h^j, c_h^j) + (\sum_{h=i+1}^{k}(p_h^j + c_h^j), 0, 0)$;
              else $(p, m, c) = (p, m, c) + \sum_{i=1}^{k}(p_i^j, m_i^j, c_i^j)$;
          $(l_j \in \mathcal{EN})$:
            if $(k \geq i)$ $(p, m, c) = (p, m, c) + \sum_{h=1}^{i}(p_h^j, m_h^j, c_h^j) + (\sum_{h=i+1}^{k}(p_h^j + c_h^j), 0, 0)$;
            else $(p, m, c) = (p, m + (i - k) * \mathcal{W}eight(T_j, w_l), c) + \sum_{i=1}^{k}(p_i^j, m_i^j, c_i^j)$;
        end case
    end
    $list = append(list, (p, m, c))$;
  end
  return $list$;
end

Figure 18: The $\mathcal{H}_{\texttt{AND}}$ algorithm

Function $\mathcal{H}_{\texttt{OR}}(r : list\_of(l : \mathcal{EN} \cup \mathcal{OP}, r : list\_of(\mathcal{PMC}))) : list\_of(\mathcal{PMC})$
begin
  Let $A = \{\#r_i \mid 1 \leq i \leq n, r_i[1] = (\bar{p}, \bar{m}, \bar{c}), \bar{c} \neq 0\}$;
  if $(A = \emptyset)$ **return** $[(0,0,0)]$; **else** $rip_{max} = \sum_{v \in A} v$;
  $plus = 0$;
  Let $r = ([\langle (l_1, T_1), r_1 \rangle, \ldots, \langle (l_n, T_n), r_n \rangle])$;
  for $i = 1$ **to** n **do if** $(l_i \in \{\texttt{AND}, \texttt{OR}\})$ $r_i = Normalize(r_i)$;
  for $j = 1$ **to** n **do**
  begin
    Let $r_j = [(p_1^j, m_1^j, c_1^j), \ldots, (p_k^j, m_k^j, c_k^j)]$ and $k = \#(r_j)$;
    if $(l_j \in \{\texttt{AND}, \texttt{OR}\})$ $plus = plus + (p_1^j + c_1^j)$;
    else $plus = plus + \sum_{h=1}^{k} (p_h^j + c_h^j)$;
  end
  for $i = 1$ **to** rip_{max} **do**
  begin
    Let $\bar{r} \in \{r_1, \ldots, r_n\}$ be the list s.t. $\bar{r}[1] = (p, m, c) = max_{\mathcal{E}}([r_1[1], \ldots, r_n[1]])$;
    $deleteHead(\bar{r})$;
    If $(i = 1)$ $list = [(plus - c, m, c)]$;
    else $list = append(list, last(list) + (-c, m, c))$;
  end
  return $list$;
end

Figure 19: The $\mathcal{H}_{\texttt{OR}}$ algorithm