Structural Similarity between XML Documents and DTDs

Patrick K.L. Ng and Vincent T.Y. Ng

Department of Computing, the Hong Kong Polytechnic University, Hong Kong
{csklng, cstyng}@comp.polyu.edu.hk

Abstract. The use of XML documents in the Internet continues to grow. Need for the analysis of XML documents from heterogeneous sources is arisen, in which documents would conform to different DTDs. In this paper, we propose a measure on the structural similarity among XML documents and DTDs, which is natural to understand and fast to calculate. The measure is defined as a weighted sum of the local measures of document elements with a weighting scheme based on their subtree sizes. While the local measure of an element is defined as its edit distance against its declaration, viewed as regular expression, in the DTD. Based on our definition, an algorithm for edit distance calculation between a string and a regular expression is proposed, which is modified from the algorithm applied in the regular expression matching problem. The advantage of the measure comes with its natural definition and linear complexity.

1 Introduction

DTD is a formal description in XML declaration syntax of a particular type of XML documents. It specifies the structures for XML documents; what names are to be used for different types of elements, where they may occur, and how they all fit together. Any document, which conforms to the specification in DTD, is called *valid* with respect to the DTD.

XML documents conforming to a DTD will have similar structure. This structure itself can also contains the semantic information, for example, a document having the tags, <portfolio>, <deal>, <maturity>, etc., is probably describing an investment portfolio. Obviously, the documents describing related contents may not have the same structure, especially when they come from different sources. Instead, they are more likely to have similar structures. This arises the interest to quantify the structural similarity so that similar structured documents can be clustered. It can be applied in document indexing as well.

In this paper, we first define the local similarity of elements using the edit distance and then aggregate the values by a weighting scheme based on the subtree sizes of the document. This approach takes into consideration the element order and the element similarity irrespective of the level. The idea is natural, easy to understand and fast to calculate, which makes it useful in both document indexing and clustering.

2 Related Work

This problem was addressed recently in [5], in which metric was defined as the weighted sum of the aggregated *PMC* (plus, minus and common tags) value against the closest instance DTD. However, the definition of *PMC* assumed that the elements were unordered but it is not true for both the XML documents and DTDs. Moreover, the *PMC* value was calculated by comparing elements from the same levels, this level restriction made all the substructures of an unmatched element ignored.

In [12], measure was defined between two XML documents based on edit distance with restricted edit operations. While the measure is target for document clustering, however, this metric cannot fully achieve our objective; namely, any two XML documents conforming to the same DTD with different degree of repetitions for an “*” operator will have distance of their degree difference, which is not zero.

In our definition, we preserve the element order in XML documents and are able to identify similarity in unmatched levels. Moreover, the more a document is conforming to a DTD, the higher the similarity value it is, in particular, highest similarity (one) is always returned when the document conforms to the DTD.

3 Local Similarity Consideration

Our idea comes from a very natural sense; if two composite objects are equivalent or similar, they should also be equivalent or similar in local sense. In the context of structural similarity, we consider the similarity at element level using edit distance, which is local in nature. This constitutes the basis of our metric.

3.1 Matching XML Documents with DTD with Errors Allowed

The main purpose of DTD is to ensure the validity of the XML documents. What’s meant by a valid XML document? It means that every elements (and attributes) in the XML document are present and conform to the structure of the hierarchical specification in the DTD. In order to make the discussion concise and clear, we focus on the elements only. The treatment of attributes is very similar so that we leave it to the reader. Figure 1 below shows an example of DTD, the corresponding valid XML document and the tree representation of the XML document.

Pick the element *deal* with id “BOND1016” from the XML document as example, it has the following child element list (*id*, *customer*, *maturity*, *schedule*, *schedule*) which matches the corresponding declaration of element *deal* in the DTD: (*id*, *customer*?, *maturity*, (*amount*|*schedule**)), in the sense of regular expression matching. Note that there is no ambiguity in which element declaration to refer to (if it ever exists) because each element is uniquely declared in the DTD. However, this does not imply that the child element, say *schedule*, will also conform to the DTD, which demands further validation against its declaration in DTD. It is necessary to go through the DTD validation process for every elements appearing in the XML document.

Fig. 1. Example of DTD (top left), XML document (top right) and the tree representation of the XML document (bottom)

Definition 1. If every elements of X (XML) conform to D (DTD), then we denote as “ X matches D ”. Then it is obviously true if X is a valid with respect to D .

We can do label substitutions, say $a=deal$, $b=id$, $c=customer$, $d=maturity$, $e=schedule$, $f=amount$, then checking of the id “*BOND1016*” can be viewed as checking whether the string “*bcdee*” matches the regular expression “*bc?d(f|e*)*”. With the setting above, matching the XML elements to the DTD can be viewed as matching the corresponding child element strings with their regular expressions in DTD.

In the previous discussion, we only consider whether an XML is valid or not. However, even a document is not valid with respect to DTD, we are still interested in how close to a valid document. For any element e in a document X , which is not necessarily valid with respect to a DTD D , it will fall into one of the three cases: (1) e matches D ; (2) e exists in D but does not matches D ; (3) e does not exist in D . If we can quantify the above cases, it can represent the local structural similarity between X and D , i.e. degree of matching at element level. In this section, we will only consider the cases (1) and (2) above while case (3) will be tackled at the end of Section 3.

In both cases (1) and (2), there exists regular expression r from D corresponding to each element e from X . We can make use of the edit distance to determine the distance between s and r . Edit distance is the metric to determine the distance between two strings; any string can be transformed into any other string by a combination of three basic operations; insertion, deletion and substitution. The edit distance is defined as the minimum number of basic operations for such a transformation. For the strings $abcdeh$ and $aaacde$, it can be shown that the three operations: (1) substitute b into a ($aaacdeh$); (2) insert a before c ($aaacdeh$); (3) delete h ($aaacde$), is a minimum transformation and hence the edit distance is 3. With this metric, we can easily extend the definition for strings and regular expressions.

Definition 2. Denote $EDIST[s_i, s_j]$ as the edit distance between two strings s_i and s_j . Then define the edit distance between a string s and a regular expression r , $EDIST[s, r]$, as $\min\{EDIST[s, s_i] : s_i \text{ matches } r\}$. If e is an element in an XML document and there exists the corresponding declaration in the DTD, we denote $EDIST[e]$ as $EDIST[s_e, r_e]$ where s_e is the child element string of e and r_e is the regular expression representing the declaration of e in the DTD. For example, $EDIST[aa, a*b] = \min\{EDIST[aa, b], EDIST[aa, ab], EDIST[aa, aab], EDIST[aa, aaab], \dots\}$, which is equal to 1, from either $EDIST[aa, ab]$ or $EDIST[aa, aab]$. Of course, the determination of edit distance becomes more complicated if the string and the regular expression are not simple.

3.2 Determination of Edit Distance

The problem of matching a string s with a regular expression r allowing errors has been widely studied, which is called the *Approximate Regular Expression Matching* problem [2,3,4,7,etc]; the problem statement is to find out all the substrings s_i of s such that $EDIST[s_i, r] \leq d$, for some given error d . In [3], it defined a function $E[]$, which is very similar our $EDIST[]$ function, as $\min\{EDIST[s_i, r] : s_i \text{ is a substring of } s \text{ ended at the last character of } s\}$.

The difference is that our $EDIST[]$ function considers only the whole string s while $E[]$ took into account of all substrings of s ended at the last character of s in the string edit distance. In this section, we will present the idea and recurrence formula for $E[]$, following similar arguments in [3] (the proof was come from [2]) for completeness. Then we will show that modifying the initial condition in the formula will make it accounting for our $EDIST[]$ function.

Definition 3. For a regular expression r , an automaton M can be constructed using the Thompson's construction [3]. Define

$Pre(i)$ as the set of predecessors of the node i in M

$\overline{Pre}(i)$ as the subset of $Pre(i)$ excluding back edges

where i is the topological numbering of nodes in M . For any string $s = s_1s_2\dots s_n$, $E[i, j]$ is defined as the minimum edit distance between any string that can reach node i in M and any substring of s that ends at s_j . If I is a set of nodes in M , denote $E[I, j]$ as $\min\{E[i, j] \text{ for all } i \text{ in } I\}$.

Definition 4. $E[i, j]$ must be equal to $E[k, j-1]$ for some $k (\leq i)$ plus the cost of editing a string w into s_j where w can range over all the strings on paths from $k+1$ to i in M . Define $E'[i, j]$ as the best edit distance over all strings w that do not take across a back edge in M .

The idea is to use recurrence to calculate $E[j]$. However, for any node i in M , the last word w may contain nodes coming from later node through some back edges, which is still not yet evaluated at that moment. Therefore, we by pass the problem by first calculating the $E'[j]$, which assumes w will not pass any back edge. By considering different cases of recurrence, we arrive at the recurrence formula (stated in [3]).

$$\text{For } j \in [0, n] \quad E[\theta, j] = 0$$

$$\text{For } i \neq \theta, \quad E[i, 0] = \begin{cases} \min E[\overline{\text{Pre}(i)}, 0] + 1 & \text{if } i \text{ is a non-}\varepsilon \text{ node} \\ \min E[\overline{\text{Pre}(i)}, 0] & \text{if } i \text{ is an } \varepsilon \text{ node} \end{cases}$$

$$\text{For } j \in [1, n] \text{ and } i \neq \theta,$$

$$E'[i, j] = \begin{cases} \min \left(E[i, j-1] + 1, E[\text{Pre}(i), j-1] + \begin{cases} 0 & \text{if } r_i = s_j \\ 1 & \text{otherwise} \end{cases} \right) \\ E'[\overline{\text{Pre}(i)}, j] \end{cases}$$

$$E[i, j] = \min \left(E'[i, j], \min \left(E'[\text{Pre}(i), j-1], E'[\overline{\text{Pre}(i)}, j-1] \right) + \begin{cases} 1 & \text{if } i \text{ is a non-}\varepsilon \text{ node} \\ 0 & \text{if } i \text{ is an } \varepsilon \text{ node} \end{cases} \right)$$

Remember that the difference between $E[j]$ and $EDIST[j]$ is whether we consider substrings in the distance definition. However, the major arguments in [3] did not make any difference if we consider the whole string rather than all the substrings ended at the last character. At a closer look at the formula above, the only difference comes from the initial condition; in $E[j]$ case, $E[\theta, j] = 0$ for all j as arisen from empty substring. For our $EDIST[j]$ case, $EDIST[\theta, j] = j$ for j between 0 and n .

This constitutes the recurrence formula for our $EDIST[j]$ function, which runs at the same efficiency as the original algorithm.

3.3 Normalization of Edit Distance

With the distance function $EDIST[j]$, we can determine how far an element in an XML document is from its declaration in the DTD. However, the edit distances for different elements in the same document may not be comparable; for example, we have two elements a, b having child strings “ c ” and “ $efghijklmn$ ” and their corresponding regular expressions “ y^* ”, “ z^* ” respectively from a DTD.

Obviously, $EDIST[a] = EDIST[c, y^*] = EDIST[c, \text{empty string}] = 1$ while $EDIST[b] = EDIST[efghijklmn, z^*] = EDIST[efghijklmn, \text{empty string}] = 10$. Intuitively, both element a and b are simply totally different from the corresponding declarations in the DTD but the edit distances show very different numbers. That arises the need to normalize the edit distances to make them comparable.

For a string s and a regular expression r , we denote $len(s)$ as the length of string s and $minlen(r)$ as $\min\{len(s); s, \text{ matches } r\}$. Then the maximum possible $EDIST[s, r]$ is $\max\{len(s), minlen(r)\}$ because s can always be converted into a string s_r matching r by inserting (or deleting) the extra length of the longer side and substituting the characters of the common length. We then normalize the edit distances by using this maximum possible distance as the denominator.

Definition 5. Define $Sim[s, r]$, the structural similarity function between a string s and a regular expression r , as $1 - (EDIST[s, r] / \max(len(s), minlen(r)))$.

If s presents a child element string of an element e in an XML document while r represents the corresponding declaration in DTD, then we define $Sim[e]$ as $Sim[s, r]$. Obviously, $Sim[e]$ is a number between 0 and 1 inclusively. Intuitively, value 1 means totally match while 0 means totally mismatch.

The $Sim[]$ function provides a mean to quantify the local structural similarity for the elements in a XML document with a DTD. In Sect. 3.1, we have not yet tackled the case (3) of those elements e existing in the document while not in the DTD. Intuitively, it is simply a total mismatch with the DTD and we can assign a value 0 to the $Sim[]$ function making the function well-defined in all cases. Then it is obvious that $Sim[e] = 1$ for all e if X is a valid document.

4 Global Similarity Consideration

Our goal is to consider the structural similarity among XML documents and DTDs as a whole instead of only locally. Therefore, we discuss in this section on how to promote the local consideration into global.

Artificial Root. In the previous section, it stated that “ $Sim[e] = 1$ for all e if X is a valid document”; however, the sufficient condition statement is not true. Consider the following XML document against the DTD described in Fig. 1. All the elements in the document match the declarations in the DTD.

```
<schedule>
  <date>16OCT2002</date>
  <amount>150000</amount>
</schedule>
```

However, the document does not conform to the DTD because a valid document should have the root element $\langle portfolio \rangle$ instead of $\langle schedule \rangle$. But why did not the matching check (by $Sim[]$) identify that? It is because the root element $\langle schedule \rangle$ has no parent element so that it did not fall into any matching check. In order to force the root element into the checking, we can add an artificial root, say R , to both the document and the DTD with the original roots as their child nodes. In other term, we effectively add an element R having the child element string $\langle schedule \rangle$, which corresponds to the regular expression $\langle portfolio \rangle$. Then we get $Sim[R] = 0$ because the $\langle schedule \rangle$ is totally different from $\langle portfolio \rangle$, which successfully detected the invalidity.

We called such XML documents and the DTDs as rooted, it is obvious that the artificial root will not affect either the values of the $Sim[]$ function or the validity of the

original documents. In other words, a rooted XML document X is valid with respect to a rooted DTD D if and only if $Sim[e] = 1$ for all elements e in X . From now on, all the XML documents and DTDs assume rooted unless otherwise specified.

Weighting Scheme. Intuitively, different elements should have different level of importance in an XML document. In other words, the bigger the subtree of an element in the document tree, the greater the importance the element should be.

Definition 6. For any element e in an XML document, $Weight[e]$ is defined as the size of the subtree rooted at e , excluding e itself. As a result, all the leaf nodes (i.e. actual data) have zero weights. Figure 2 below shows the element weights for the XML example in figure 1, which has total weight (for all elements) of 99.

Fig. 2. Element Weights (left) and the weight distribution (right)

We can make use of the element weights to define a weighting scheme, as a consequence, the greater the substructure of an element, the greater the weighting percentage. Then we can gather local pieces of similarity information to give a global view, which takes care of the semantic meaning for XML documents.

5 Structural Similarity

Now, we can define the structural similarity as the weighted sum of local similarity ($Sim[I]$) of the elements in an XML document X with respect to a DTD D .

$$Sim[X, D] = \frac{\sum_{e \in X} Weight[e] \times Sim[e]}{\sum_{e \in X} Weight[e]}$$

Hence X is valid with respect to D if and only if $Sim[X, D] = 1$. On the other extreme cases, say X has no common element with D , $Sim[X, D] = 0$. The figures below show an example of an XML document and a DTD, and illustrate the similarity function evaluation.

Fig. 3. Evaluation of similarity function

From the above figure, we can calculate the similarity as

$$Sim[XML2, DTD2] = \frac{8 \times 1|_R + 7 \times \frac{2}{3}|_a + 2 \times 0|_b + 1 \times 1|_h + 1 \times 1|_e + 1 \times 1|_g}{8|_R + 7|_a + 2|_b + 1|_h + 1|_e + 1|_g} = 0.78$$

The definition of the similarity function is natural and simple, hence the evaluation is correspondingly straightforward. The following are the main steps for the calculation: (1) Calculate the weights of every elements in the document (hence the total weight); (2) Calculate edit distances and the maximum possible edit distances of every elements with their declarations in the DTD; (3) Calculate the weighted sum.

The part (1) above can be done by traversing the XML document tree once according to the postorder numbering. It is due to the property that the postorder numbering always goes through all the child nodes before parent nodes for every subtree in a tree. The weights can be obtained by accumulating the weights of child nodes in every subtrees.

For part (2), we can pre-build the automata for every declaration in the DTD indexed by the declaring elements, as the same automata maybe referred by many instances of the elements in the document. Then the edit distances can be calculated by the recurrence formula shown in the previous section. While the maximum possible edit distances can be much easier to calculate by counting both the lengths of the strings and the regular expressions.

By gathering the results in part (1) and (2), part (3) can be computed accordingly. The time complexity in part (2) dominates the other two parts. The time complexity of the overall algorithm is $O(R_d N_x)$, where R_d is longest $minlen(r)$ among all r in D and N_x is the size of X , which is fast in practice. The algorithm is shown below.

Fig. 4. Structural similarity algorithm

6 Experiment Results

The proposed similarity measure has been implemented in Java using the DOM libraries [8, 9]. In order to evaluate the accurateness and effectiveness of the measure and the algorithm, both synthetic and real data are evaluated.

Synthetic Data: Synthetic data is generated by an XML document generator [13], which generates valid documents for a chosen DTD. An error rate can be configured so that generated documents will not fully conform but closely conform to the DTD, depending on the error rate specified. 3 sets of 500 documents were generated this way with different error rates: (A) No error, (B) 10%, (C) 50%. Similarity measures of the 3 datasets are run against the chosen DTD and shown below.

Similarity Range	Data Set A	Data Set B	Data Set C
$\text{Sim} = 1.0$	500	0	0
$0.8 \leq \text{Sim} < 1.0$	0	166	0
$0.6 \leq \text{Sim} < 0.8$	0	237	14
$0.4 \leq \text{Sim} < 0.6$	0	25	114
$0.2 \leq \text{Sim} < 0.4$	0	58	26
$0.0 \leq \text{Sim} < 0.2$	0	14	346
TOTAL	500	500	500

It verified that the similarity measure always generates the value 1 for valid documents, while < 1 for invalid documents. Moreover, the majority of data sets B and C are considered as “good but not exact matched” (403/500 between 0.6 and 1.0) and “badly matched” (486/500 between 0.0 and 0.6) against the DTD respectively, hence it successfully measures the similarity against a DTD.

Real Data: Real data are downloaded from the Internet to evaluate the effectiveness of the measure. 727 XML documents are retrieved from a web site [6], generated by a common DTD. 2 documents are selected randomly from the population to generate a DTD based on the XTRACT system mechanism (Generalization, Factoring and MDL

Principle) [10], which appeared to be about 10% different from the original DTD. Then the similarity values of all the documents against both the original and generated DTD are calculated and listed below.

Similarity Range	Original DTD	Generated DTD
$\text{Sim} = 1.0$	727	339
$0.8 \leq \text{Sim} < 1.0$	0	58
$0.6 \leq \text{Sim} < 0.8$	0	335
$0.0 \leq \text{Sim} < 0.6$	0	0
TOTAL	727	727

Even the generated DTD was based only on a small portion (2 out of 727) of the documents, it still showed “good” similarity (≥ 0.6) against the whole population.

7 Conclusion

In the paper we propose a metric for the structural similarity between XML documents and DTD. The metric takes care of the semantic meaning of XML documents, which is easy to understand. We also present an algorithm, which is fast to run in practice. With carefully chosen DTD among an arbitrary set of XML documents, this metric can be used for XML clustering, classification and indexing, which constitutes the future direction of the project.

References

1. Zhang, K., Shasha, D.: Simple Fast Algorithm for the Editing Distance Between Trees and Related Problems, *SIAM J. COMPUT* 18(6) (1989) 1245–1262
2. Myers, E. W., Miller, W.: Approximate matching of regular expressions, *Bull. Math. Biol.*, 51 (1989) 5–37
3. Wu, S., Manber, U., Myers, E.: A Subquadratic Algorithm for Approximate Regular Expression Matching, *Journal of Algorithms*, 19 (1995) 346–360
4. Myers, E. W.: A four-Russian algorithm for regular expression pattern matching, *J. Assoc. Comput. Mach.* 39(2) (1992) 430–448
5. Bertino, E., Guerrini, G., Mesiti, M., Rivara, I., Tavella, C.: Measuring the Structural Similarity among XML Documents and DTD, Technical Report DISI-TR-02-02, Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova (Dec 2001)
6. ACM SIGMOD Record: XML Version, <http://www.acm.org/sigmod/record/xml/>
7. Navarro, G.: A guided tour to approximate string matching, *ACM Computing Surveys* 33(1), (2001) 31–88
8. W3C, Document Object Model (DOM) (1998)
9. The Apache XML Project, <http://xml.apache.org/>
10. Garofalakis, M., Gionis, A., Rastogi, R., Seshadri, S., Shim, K.: XTRACT: A System for Extracting Document Type Descriptors from XML Documents, *Proceedings of ACM SIGMOD* (2000)
11. Schlieder, T., Naumann, F.: Approximate Tree Embedding for Querying XML Data, *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval* (2000)
12. Nierman, A., Jagadish, H. V.: Evaluating Structural Similarity in XML Documents, *Fifth International Workshop on the Web and Databases (WebDB 2002)*
13. XML at Sun, <http://www.sun.com/software/xml/>