Finding Syntactic Similarities Between XML Documents

Davood Rafiei
University of Alberta
drafiei@cs.ualberta.ca

Daniel L. Moise
University of Alberta
moise@cs.ualberta.ca

Dabo Sun
University of Alberta
dabo@cs.ualberta.ca

Abstract

Detecting structural similarities between XML documents has been the subject of several recent work, and the proposed algorithms mostly use tree edit distance between the corresponding trees of XML documents. However, evaluating a tree edit distance is computationally expensive and does not easily scale up to large collections. We show in this paper that a tree edit distance computation often is not necessary and can be avoided. In particular, we propose a concise structural summary of XML documents and show that a comparison based on this summary is both fast and effective. Our experimental evaluation shows that this method does an excellent job of grouping documents generated by the same DTD, outperforming some of the previously proposed solutions based on a tree comparison. Furthermore, the time complexity of the algorithm is linear on the size of the structural description.

1 Introduction

There is a large and still growing number of applications that use the eXtensible Markup Language (XML) for data exchange. Storing documents generated by two or more applications in a database can be challenging as data may not conform to a non-trivial unifying DTD[1] or the unifying DTD may be too complicated. Under these conditions, it may not be efficient to construct a single DTD or to use one specific relational mapping to store all documents. This more of an issue for documents found on the Web due to the heterogeneous nature of the environment and the applications that generate the documents. A possible solution is to group documents that conform to the same or similar DTDs together before storing them.

Grouping similar XML files together can lead to a better storage mapping and indexing [9, 13, 15]. This in turn can improve the efficiency of the retrievals; for two elements that are under structurally similar paths, it is more likely that either both elements or none satisfy a given query, compared to the case where the elements are selected arbitrarily. Without a proper grouping, elements under structurally similar paths can be scattered at different locations in the storage device, thus making the retrievals inefficient.

Another application that can benefit from clustering XML documents is effective DTD extraction [11]. If structurally similar XML documents are clustered before a DTD extraction, more specific and accurate DTDs can be constructed for documents in each cluster. A more accurate DTD can be useful in query evaluation and can limit the access to only the relevant portions of data; this will in turn increase the efficiency.

The problem to be addressed is how to define the similarity between XML files, i.e. what kind of metric should be used, such that two documents generated by the same DTD are grouped together. In this paper, we take a data-centric view of an XML document and do not make use of the order in which the elements appear in the document. This view of data is quite useful in query processing and has contributed much to the success of relational database systems.

We evaluate the effectiveness of our method and our parameter settings by running experiments over both real and synthetic data. The results show that our method outperforms previously proposed alternatives based on tree comparison, doing a better job of grouping documents generated by the same DTD. We also outline possible limitations of our approach and suggest some solutions to overcome those limitations.

The paper is organized as follows. Related work is reviewed in Section 2. Our notion of structural summary and our approach for comparing two XML documents are presented in Section 3. Section 4 presents our experimental evaluations. Section 5 discusses the results and concludes the paper.

2 Related Work

Since the structure of an XML document can be described as a tree or as a graph (in general), related work

[1]It is always possible to construct a unifying DTD for a set of XML documents, but a trivial DTD that accept many possible documents would not be much useful.

includes the past work on matching trees and graphs. If trees can be treated as ordered, i.e. changing the order of the elements changes the structure of a document, there is considerable work on finding the edit distance between ordered trees which are applicable [14, 21, 3, 16].

There has been recently more specific work on finding the structural similarity between XML documents, based on the general assumption that the structure of an XML document is described as an ordered tree [11, 5, 23, 20]. In particular, Nierman and Jagadish propose a few tree edit operations and a dynamic programming algorithm to find the structural similarity between XML documents [11]. They run experiments to show that using the edit distance with the new set of operations does a better job of clustering the documents generated by the same DTD, compared to some earlier approaches [21, 3]. Improvements are reported over Zhang and Shasha's tree edit distance [21] when the weights of the edit operations are learned from a training dataset [23]. In the presence of a training dataset, Zaki and Aggarwal propose a rule-based classifier that relates frequent ordered tree structures in an XML document to class labels [20].

Again making use of the order of the elements, Flesca et al. propose an interesting approach for detecting the structural similarity between XML documents [5]. Their approach is based on the idea of encoding the structure of an XML document as a time series in which an impulse represents the occurrence of a tag. They use the Euclidean distance between some of the Fourier descriptions of these series to find out if two documents are similar.

There is past work that adopt a data centric view of XML documents and use parent-child tags and twigs for clustering or classification. In particular, Theobald et al. [18] use parent-child tags, pairs of tags and content terms, twigs and the semantic relationships between terms (defined by Wordnet [19]) to classify each XML document into one of a few known classes. A hierarchical clustering algorithm based on common parent-child tags between documents is proposed by Lian et al. [10]. Consider this algorithm applied to two paper elements shown in Figure 1. Despite the structural similarity that exists, the elements have no common parent-child tags and will be considered non-similar.

Bertino et al. propose a metric for finding the similarity between the structure of an XML document and a DTD, taking into account both the tag names and the structural descriptions [1]. Based on the same idea, they develop a notion of similarity between two XML documents, but they report no experimental results.

Finally, the problem of finding the edit distance between two unordered trees is shown to be NP-complete [22], so is the problem of subgraph isomorphism [6]. No polynomial-time algorithm is known for graph isomorphism, and neither is it known to be NP-complete.

```
<paper>
    <journal>
        <author>...</author>
        <title>...</title>
        <year>...</year>
    </journal>
</paper>
<paper>
    <conference>
        <author>...</author>
        <title>...</title>
        <year>...</year>
    </conference>
</paper>
```

Figure 1. Two XML documents

3 Structural Sketch and Similarity

An XML document can be modeled as a node-labeled directed tree [2] where each node in the tree represents either an element or an attribute in the corresponding XML document. When a node represents an element, the node is labeled with the tag name of the element and when a node represents an attribute, it is labeled with the name of the attribute. Each edge of the tree represents a hierarchical inclusion relationship between either two elements or an element and an attribute. Since we are only interested in the structure of an XML document, we ignore other possible node types such as data values, comments and processing instructions. Furthermore, to make our description more concise, we require that each path starting from the root and ending at a leaf appears at most once. We refer to this tree description of an XML document as *structure tree* [3].

Definition 1 A *structure tree* for an XML document *d* is a tree *t* such that for every path in *d* there is a corresponding path in *t* and vice versa and *t* is minimal, meaning that no edges or nodes in *t* can be removed while still preserving the same relationship.

As an example, an XML document and a structure tree of the document are shown in Figure 2.

Given two XML documents, we want to find out if the two documents are structurally similar irrespective of the order in which the elements appear in each document. Since the structure of an XML document can be described as a structure tree, a solution is to compare the respective structure trees of the two XML documents. However, an XML

[2]Two special attributes ID and IDREF, when present, may not be properly represented in a tree.

[3]Our definition of *structure tree* is very similar to the concept of dataguide in the OEM model [7].

```
<people>
    <person>
        <name>Tom</name>
        <address>UofA</address>
    </person>
    <person>
        <name>Cat</name>
        <age>18</age>
    </person>
</people>
```

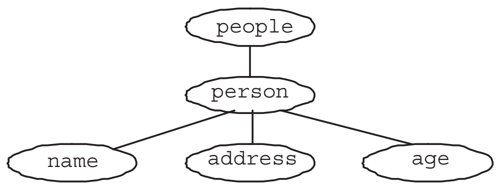

Figure 2. An XML document and a structure tree of the document

document can have multiple structure trees and it is not clear which one should be used for a comparison. Furthermore, the problem of finding the edit distance between two unordered trees is not easy.

Our proposed solution is to describe the structure of an XML document as a set of paths and to avoid a tree matching problem. Even though an XML document can have more than one structure tree, they all have the same set of paths in common with the document (this is directly inferred from the definition of a structure tree). Given an XML document, all paths can be extracted within one scan of the document. To make our description more concise, we are interested in every path that starts from the root and ends at a leaf. These paths here are referred to as the *root paths*.

Definition 2 A *root path* for an XML document is a path $a_1a_2\ldots a_h$ in its respective structure tree where a_1 is the root node, a_h is a leaf and for every a_ia_{i+1} in the path there is a direct parent/child relationship.

For instance, *people/person/name*, *people/ person/address*, and *people/person/age* are the root paths of the XML document shown in Figure 2.

To account for similar but not identical paths between two XML documents, we use in our comparisons not only the root paths but also all subpaths of the root paths (i.e. a consecutive sequence of tag names).

Definition 3 The *path set* of an XML document is the union of all of its root paths and all subpaths of the root paths.

The path set of the document shown in Figure 2, for instance, is: $\{$ *people/person/name*, *people/ person/address*, *people/person/age*, *people/person*, *person/name*, *person/address*, *person/age*, *people*, *person*, *name*, *address*, *age* $\}$.

Furthermore, to count for paths that appear more frequently, we further retrieve from each XML document the frequency of each path.

Given the features of an XML document as a set of *(path, frequency)*, we can use standard set comparison techniques to find out if two documents are similar; the equality operation between two paths is the standard case-insensitive comparison of two strings. Informally, we call two XML documents similar if a large fraction of the paths in their path sets are the same. There are a number of ways of computing such an overlap including the Jaccard Coefficient or its extensions, the Dice Coefficient and the well-known *Cosine* measure (see an information retrieval text such as [12] for more details). Given the pair-wise similarity between all documents in the collection, a clustering algorithm can be applied to group similar XML documents into clusters, where every cluster should ideally represent a set of XML documents that share the same DTD.

Given a document with n root paths, each of length l, there are $nl(l+1)/2$ possible subpaths. A similarity comparison between two documents can be done within one scan of their path sets [4]. With the path set size set to $nl(l+1)/2$, the time complexity in terms of the number of string comparisons for our approach is expected to be $O(nl^2)$, compared to $O(n^2l^2)$ or higher node comparisons reported for a tree edit distance computation.

4 Experiments

In order to evaluate the effectiveness of our approach we ran experiments against both real and synthetic data. Our real data, referred to here as *RE*, was the online XML version of the ACM Sigmod Record from March 1999 [17]. This collection contained XML files shared among four DTDs: *ProceedingsPage*, *IndexTermsPage*, *OrdinaryIssuePage* and *SigmodRecord*. The collection had 989 XML files with a total size of 3.35 MB. The breakdown of the number of XML files per each DTD was as follows:

17 XML files	for *ProceedingsPage*
920 XML files	for *IndexTermsPage*
51 XML files	for *OrdinaryIssuePage*
1 XML file	for *SigmodRecord*

[4]The exact algorithm may vary depending on the similarity function used, but one can use hashing to identify the matching pairs of paths between two sets. Assuming that the data is distributed evenly by the hash, each path needs to be compared to not more than a few of other paths

We used the Oracle parser [4] to extract paths from XML documents. We observed that the XML parser found some errors for three of the XML files from the Sigmod collection: “00969.xml”, “01200.xml” and “01446.xml”. When we checked the XML files for the source of the error, we found out that the character “&” appeared as text. But, the XML processors interpreted the character “&” as the start of an entity reference. We modified these three files by replacing the character “&” with “&*amp;*”.

For synthetic data, we selected all DTDs reported by Nierman and Jagadish [11] except one called *HealthProduct.dtd* which we couldn't obtain. This set of DTDs here are referred to as *DTD set A*. To further test our method, we also used an extended set of DTDs which included *DTD set A* and 5 additional DTDs. We refer to this set of DTDs as *DTD set B*[5] Using the IBM XML generator [8], we generated 100 XML files for each DTD and each setting of the parameters *M* (the maximum repeat of an element marked with a ‘+’ or ‘*’) and *P* (the probability that an optional attribute appears). We grouped the files into eight data sets:

DS1: M=4, P=0.75, *DTD set A*
DS2: M=4, P=1, *DTD set A*
DS3: M=8, P=0.75, *DTD set A*
DS4: M=8, P=1, *DTD set A*
DS5: M=4, P=0.75, *DTD set B*
DS6: M=4, P=1, *DTD set B*
DS7: M=8, P=0.75, *DTD set B*
DS8: M=8, P=1, *DTD set B*

Our dataset was deliberately chosen as a superset of the datasets used by Nierman and Jagadish [11] so that we could compare our results not only with those of Nierman and Jagadish but also with the results of the algorithms suggested by Chawathe [2] and Shasha et al. [16] without implementing these algorithms. Note that our synthetic data is generated randomly but using the same parameters as in [11], so the two datasets may not be identical.

4.1 Evaluation

Ideally, we want to have all documents conforming to the same DTD be clustered together, but in practice this may not be the case. To measure the effectiveness of each method, we use the same notion of mis-clustering introduced by Nierman and Jagadish, i.e. the minimum number of the documents that can be moved in order to obtain all documents conforming to the same DTD in the same cluster. The hierarchical agglomerative clustering algorithm in the R project for statistical computing[6] was used to cluster our data sets. The result of the clustering is a dendrogram, showing the clusters that collapse in each step.

We use the number of mis-clusterings reported by Nierman and Jagadish (as shown in Table 1) over data sets 1-4 and the Sigmod collection as a base line to compare the performance of our approach against theirs and those of Chawathe [2], Shasha [16] and Tag Frequency [11].

	RE	DS1	DS2	DS3	DS4
Nierman	0	10	2	11	9
Chawathe	3	16	8	30	25
Shasha	3	16	9	32	39
Tag Frequency	3	22	21	35	40

Table 1. Number of mis-clusterings reported by Nierman and Jagadish

Table 2 shows the number of mis-clusterings obtained using our approach when each document is represented as a binary vector (BV), a frequency vector (FV) and a normalized frequency vector (NFV) of the path occurrences. The *Cosine* measure used to final the similarity between two documents. Using the path frequencies improves the clustering accuracy. We also did similar experiments using both the Jaccard and the Dice Coefficients. The results were either comparable or worse than the Cosine measure, thus it was not reported.

One question is if we really need to extract paths and if we can obtain similar results by only using the tag frequencies and perhaps with some variations of the similarity measure. Since the Cosine measure performs the best in our experiments, we choose this measure for tag frequencies. We also pick two additional distance functions, namely the city-block distance because it is previously used [11] as reported in Table 1 and the Euclidean distance. Table 3 shows that the tag frequency is not enough to compare the structural similarity between XML documents.

We found some inconsistencies between our results for the tag frequency and the results reported earlier using the city-block distance [11]. One possible explanation is that we may be counting the number of mis-clusterings differently. In our case, the number of mis-clusterings are counted manually, but we are not sure how this is done in

	RE	DS1-DS4	DS5	DS6	DS7	DS8
BV	0	0	33	30	25	29
FV	0	0	0	0	0	0
NFV	0	0	0	0	0	0

Table 2. Number of mis-clusterings using our approach

[5]All these DTDs are available online at `http://www.cs.ualberta.ca/~drafiei/dtds.html`.
[6]`http://www.r-project.org`

	RE	DS1	DS2	DS3	DS4
City block	24	208	200	211	240
Euclidean	24	0	62	0	0
Cosine	68	38	33	39	35

Table 3. Number of mis-clusterings using the tag frequency

the earlier reporting. Since our counts of the mis-clusterings are higher than those reported by Nierman and Jagadish, we feel justified to say that we are over-estimating the number of mis-clusterings. Thus, our approach outperforms previously-proposed methods.

5 Discussions

We have proposed a simple and yet efficient approach to find the structural similarity between XML documents. We have also evaluated our approach with various data sets. On a modest hardware (Pentium 4, 2.8GHz CPU and 1GB RAM), the path extraction (in Java) for the Sigmod collection with 989 documents took 20 seconds. Computing the pair-wise distances (in C++) between all these documents took only 98 seconds.

A limitation of our approach is when we want to detect a similarity between documents with the same structures but different tag names. We expect this to be less problem with an increasing use of namespaces and also the tendency to use the same tag name to refer to the same concept. However, one solution is to allow users to specify some relabeling rules. For instance, if the tag names in one document are in French and the tag names in another document are in English, a possible relabeling can be a word-to-word translation of the tag names.

Acknowledgments

This work is supported by Natural Sciences and Engineering Research Council of Canada.

References

[1] E. Bertino, G. Guerrini, and M. Mesiti. Matching an XML document against a set of DTDs. In *Proceedings of the ISMIS Symposium*, pages 412–422, 2002.

[2] S. S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the VLDB Conference*, pages 90–101, Edinburgh, 1999.

[3] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the SIGMOD Conference*, pages 493–504. ACM Press, 1996.

[4] O. Co. Oracle xdk for java. `http://otn.oracle.com/tech/xml/xdk/software/prod/xdk_java.html`.

[5] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between XML documents. In *Proceedings of the WebDB Workshop*, Madison, June 2002.

[6] M. Garey and D. Johnson. *Computers and interactability: a guide to the theory of NP-Completeness*. Freeman and Company, 1979.

[7] R. Goldman and J. Widom. Dataguides: enabling query formulation and optimization in semistructured databases. In *Proceedings of the VLDB Conference*, pages 436–445, 1997.

[8] IBM XML Generator. `http://www.alphaworks.ibm.com/tech/xmlgenerator`.

[9] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploiting local similarity for indexing paths in graph-structured data. In *Proceedings of the ICDE Conference*, pages 129–140, 2002.

[10] W. Lian, D. W. Cheung, N. Mamoulis, and S. M. Yiu. An efficient and scalable algorithm for clustering xml documents by structure. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):82–96, 2004.

[11] A. Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the WebDB Workshop*, Madison, June 2002.

[12] G. Salton. *Introduction to Modern Information Retrieval*. MacGraw Hill, 1983.

[13] H. Schöning. Tamino - a DBMS designed for XML. In *Proceedings of the ICDE Conference*, pages 149–154, 2001.

[14] S. M. Selkow. The tree-to-tree editing problem. In *In Information Processing Letters, 6(6)*, pages 184–186, 1977.

[15] T. X. Server. Frequently asked questions. `http://www.xmlstarterkit.com/tamino/faq.htm#FAQ34`.

[16] D. Shasha and K. Zhang. Approximate tree pattern matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[17] Sigmod Record in XML. `http://www.acm.org/sigmod/record/xml/index.html`.

[18] M. Theobald, R. Schenkel, and G. Weikum. Exploiting structure, annotation and ontological knowledge for automatic classification of xml data. In *Proceedings of the WebDB Workshop*, 2003.

[19] Wordnet:. a lexical database for the english language. www.cogsci.princeton.edu/ wn.

[20] M. J. Zaki and C. C. Aggarwal. Xrules: an effective structural classifier for xml data. In *Proceedings of the KDD Conference*, pages 316–325, 2003.

[21] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[22] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

[23] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity metric for xml documents. In *FGWM Workshop on Knowledge and Experience Management*, 2003.