This paper was presented at : **International Conference on Emerging of Networking, Communication and Computing Technologies (ICENCCT 2014)** Co-jointed **with International Conference on Emerging Trends of Computer Science with Educational Technology (ICETCSET 2014)** – Zurich, Switzerland on February 22-23, 2014 – Organized by : **Institute of Information System and Research Center (IISRC)**

A Fuzzy Similarity Measure for XML Documents

Seyyed-Mohammad Javadi-Moghaddam
Stefanos Kollias
National Technical University of Athens
Image, Video and Multimedia Systems Laboratory
P.C. 15780, Zographou, Athens, Greece
javadimo@central.ntua.gr, stefanos@image.ntua.gr

Abstract*:*

The growth of the Internet has resulted in the rapid growth of using XML for data representation and exchange over the Web. Finding the similarity of XML documents is a significant research task in order to effectively control and retrieve information over the web. In this paper, we propose a new approach for determining similarity of XML documents by considering their content and structure. The similarity is computed by using the Sorensen–Dice's coefficient and fuzzy intersection. We experimentally demonstrate the accuracy of the similarity method using real data sets.

Keywords-component; *XML document similarity, fuzzy set, string matching.*

I. INTRODUCTION

With the continuous growth of the Internet, Extensible Markup Language (XML) has become more popular as a dominant standard for the representation and exchange of data over the Web. This has led to massive collections of XML data and has imposed an enormous opportunity and challenge for grouping XML documents based on their context and structure. Clearly, the computation of similarity plays an important role for grouping and clustering XML documents.

XML documents are comprised of nested elements. XML Documents can be labeled as a tag tree of element nodes that each XML element is represented by a node in the tree; the node is labeled by the name or value in the XML element. Element-sub element relationships are represented by edges in the tree. This structural information is important criteria to compute the similarity. On the other hand, the value and meaning in the nodes data are significant for measuring the similarity. Therefore, to compare two XML documents, it is necessary to consider both the structure and the contents of them.

In this paper, we propose a new method for comparing two XML documents based on their structure and contents. The Sorensen–Dice's coefficient is used for computing the similarity of documents' structure. In addition, similarity of documents' contents is estimated by fuzzy analysis taking into account the names and positions of elements.

The rest of the paper is organized as follows. In section II, we review related work about finding similarity of XML documents. An approach mapping XML to tree structure is proposed in Section III. Section IV shows a

one-to-one sequencing method to transform the trees to strings. String matching and the algorithms implementing it are presented in section V. In section VI, we describe the fuzzy similarity computation. Section VII presents the experimental work. Finally, conclusions are given in chapter VIII.

II. RELATED WORK

Over the past years, there have been proposed a large amount of XML document similarity estimation methods. Some approaches have focused on similarity of documents' structure [1]. Other researchers [2] have computed the similarity based solely on the content of documents. Since structure and content of documents play an important role in assessing the similarity, the approaches based on only one of them do not provide very good results. Consequently, several authors have provided similarity algorithms based on both structure and content of XML documents [3].

In most of structure-similarity based approaches, structure could be either a labeled tree corresponding to the original structure of the XML document (the whole structure of document), or a rooted ordered labeled tree summary. For example, the XML tree can be decomposed into path information called node paths, i.e., ordered sets of nodes from the root node to a leaf node. The most usual distance measure for tree-structured data is the tree edit distance [4]. In this method, the edit distance is computed by considering alternative sequences of edit operations that can transform one object into the other. The cost of the operations in each sequence is considered, and the lowest cost sequence among these defines the edit distance between the two objects. However, computing the tree edit distance can be very expensive both in terms of CPU cost and disc I/Os.

To compute similarity based on content, we should take into account the semantics of the elements. In most cases the mapping is generated by only stand-alone XML documents, without DTD or XML schema [2].

Similarity based on both structure and content can be computed by considering the semantics of elements and nested structures of XML documents [5]. Such a method measures the similarity between XML documents by considering their structures and content, using a three-layer matching: element matching, path matching and document matching. This model is based on the bag of tree paths; it can, however, lead to a high time complexity.

As mentioned before, edit distance is time-consuming and the similarity result may not be accurate in terms of semantics. For this reason, it has been proposed to measure similarity according to string matching. Li [6] has proposed to transform tree structured data into strings with a one-to-one mapping. He has proven that the edit distance of the corresponding strings forms a bound for the similarity measures between trees, including tree edit distance, largest common sub trees and smallest common super-trees. Navarro [7] gives a good overview of the edit distance for strings and its variants. Ukkonen [8] introduces the q-gram distance as a lower bound for the string edit distance. The q-gram distance between two strings is based on the number of common sub strings of length q. Gravano et al. [9] present algorithms for approximate string joining based on edit distance and using q-grams for filtering.

In this paper, we propose a new method for measuring the similarity between two XML documents in terms of their structure and content. To achieve a high method performance, we transform the trees into strings with a one-to-one mapping. Then, the similarity of documents' structure is found by simple string matching and that of documents' content is found by simply taking into account the names and positions of elements. The overall algorithm runs in linear time with respect to the combined size of the two documents involved in the evaluation.

III. MAPING XML TO TREE

A. Ordered Labeled Tree for XML Documents

An XML document can be mapped to an ordered labeled tree where each node in the tree describes either an element or an attribute in the XML document [10]. The edges represent a hierarchical relationship between nodes that can relate two elements or an element to an attribute. They are also labeled by tag name of the element or by the name of the attribute (Fig. 1).

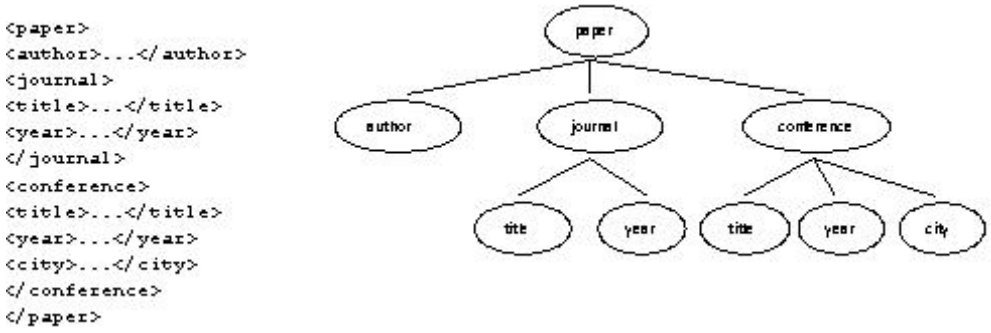

Figure 1. An XML document for publicaation

B. Level Labeled Tree for XML Documents

Level labeled tree is an ordered labeled tree with the difference that the label of a node is the number of its level. Each edge of the tree represents a hierarchical inclusion relationship between either two elements or an element and an attribute. The root's level is 0 (Fig. 2).

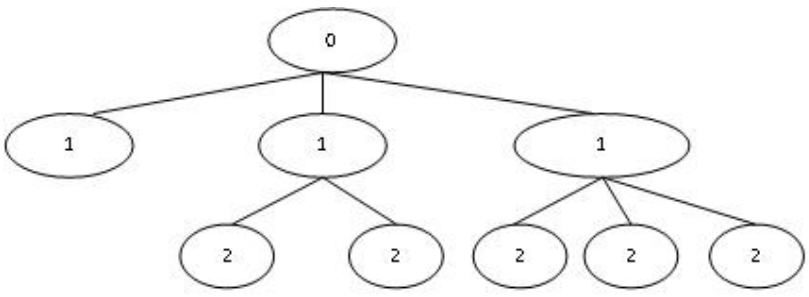

Figure 2. Level labeled tree for the XML document shown in Fig. 1

C. Weighted Tree

In order to compare two XML documents not only should we consider the nodes' names similarity but also notice the position of the nodes in the mapped XML tree [3]. To do it, we construct a tree each node of which corresponds to a node in the ordered labeled tree and is labeled with proper weight. Each edge in this tree indicates inclusion of the node, corresponding to the child one, in the node, corresponding to the parent one, within the ordered labeled tree. In this paper, we assign weight 1 for the root node. In order to reflect a path to the weight criterion, a parent node gains more weight than a child node. A child node's weight is considered equal to the parent node's weight divided by m if the parent has m ($m \geq 2$) child nodes. But if a parent has only one child, then the child node's weight becomes a half of parent node's weight in order to consider a path.

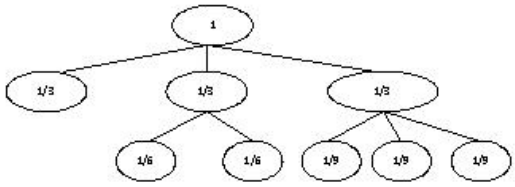

Figure 3. Weighted tree for XML document Fig. 1

IV. TREE 'S STRINGS

This section introduces a one-to-one sequencing method to transform the trees to strings. Prufer proposed a method that computes a one-to-one correspondence by removing nodes from the tree one at a time [11]. Another way to transform the tree to string is built by traversing on the tree. In this paper, we build string for the tree by using depth first algorithm. To save the relationship between the nodes, we make three strings according to the meaning and position of the nodes.

A. Depth First Search (DFS)

One of the systematic methods of visiting the vertices of a graph is DFS. Given that we are currently visiting vertex *u*, in the next step, we visit a vertex adjacent to *u* which has not yet been visited yet. If no such vertex exists then we return to the vertex visited just before *u* and the search is repeated until every vertex in that component of the graph has been visited. Time complexity of the initialization part of DFS is $\Theta(n)$, as every vertex must be visited once. Moreover, $\Theta(m)$ is time complexity of the main(recursive) part of the algorithm. In total, the algorithm's time complexity is $\Theta(m + n)$ where n shows the number of nodes that are discovered for the first time and m is the number of nodes that are visited for the second time.

B. Element String (ES)

The ES string is built by applying the DFS algorithm on the ordered labeled tree. This string compromises the objects that express semantics of the nodes. Below is an example according to Fig. 2:

ES: "paper,author,journal,conference,title,year,tile,year,city"

C. Numeral String (NS)

To build the NS, we should execute the DFS algorithm on a Level labeled tree, using a specific number for every level. By going down the tree, the number increases. According to NS, the position of each node can be defined. The NS for Fig. 2 can be computed as follows: NS: "011122222"

D. Weight String (WS)

By executing DFS algorithm on the weight tree, we can build The WS. Indeed, this string shows the amount which the nodes can be influent by root. We have used this amount as a coefficient for similarity. In the following, you can see the WS for Fig. 3. WS: "1, 1/3, 1/3, 1/3, 1/6, 1/6, 1/9, 1/9, 1/9"

V. SIMILARITY BETWEEN STRINGS

A. String Edit Distance

Similarity between strings is usually evaluated by string edit distance. String edit distance operations include substitution, insertion, and deletion of a character with the cost of each operation being always assigned to one. Many researchers have proposed approaches to compute similarity search and similarity join based on edit distance over textual data [9]. Gravano et al. [9] proposed *q*-grams to measure similarity match on textual data. A *q*-gram is a contiguous substring that has length *q*. In this approach, similarity is computed by using common *q*-grams. To improve the performance Li et al. [12] proposed a new technique called VGRAM that selects the high-quality grams from a collection of strings. To facilitate similarity searches, in [13]. Indexing structures and specific merging algorithms were proposed.

B. String Matching

String matching can be done by two methods: accurate matching and approximate matching (fuzzy string searching). Accurate string matching is used when a query string is similar identically to another string. Approximate String Matching Algorithm (fuzzy string matching) is the technique of finding strings that match approximately (rather than exactly). In this paper, we compute string similarity by using approximate string matching. To estimate string matching, we apply The Sorensen–Dice's coefficient.

1) Sorensen- Dice's Coefficient

The Sorensen–Dice's coefficient is a statistical entity used for comparing the similarity of two samples. It is computed according to the number of common species between the two samples and to the number of spices in the two samples. Given that A and B are the numbers of species, the similarity of two samples be calculated as follows:

$$Sim = \frac{2|A \cap B|}{|A| + |B|} \qquad (1)$$

C. Similarity Measure

Dice's coefficient can be used to measure two strings' similarity [14]. To do this, we need to calculate the bigrams of the two strings. A bigram is every sequence of two adjacent elements in a string of tokens, which are typically letters, syllables, or words. Taking the string "football", the set of bigrams would be {"fo", "oo", "ot", "tb","ba", "al","ll"}. The similarity of two strings is given by the formula:

$$\mathrm{Sim}(x,y)=\frac{2n_t}{n_x+n_y} \qquad (2)$$

where n_t is the number of character bigrams found in both strings, n_x is the number of bigrams in string *x* and n_y is the number of bigrams in string *y*. For example, to calculate the similarity between," table" and "taken" , we would find the set of bigrams in each word:

{ta,ab,bl,le},{ta,ak,ke,en}

Each set has four elements, and the intersection of these two sets has only one element: ta. Inserting these numbers into the formula, we calculate, Sim = (2 · 1) / (4 + 4) = 0.25.

VI. A FUZZY SIMILARITY ALGORITHM FOR XML

In this algorithm, we obtain the similarity of two XML documents according to structure and content of them. The similarity of documents' structure is found by simple string matching and that of documents' content is found by computing weights that take into account the names and positions of elements. First, two corresponding ordered labeled trees are built by the two XML documents. The nodes of these trees are the included weights and the level numbers beside the names of elements; they are computed at the same time. Next, three strings of the tree's nodes are obtained as mentioned in the previous chapters. The similarity is computed as follows:

$$FSim(d_x,d_y)= CSim(d_x,d_y) * W_c + SSim(d_x,d_y) * W_s \quad (3)$$
$$W_c+W_s=1,$$

where CSim is a content similarity, SSim is structure similarity, W_c and W_s are coefficients that define the importance of content similarity and structure similarity respectively. These coefficients are adjusted by the user.

A. Structure Similarity (SSim)

The structure similarity between two XML documents *d1* and *d2* can be calculated by the corresponding trees of *d1* and *d2*. In this approach, the structure of a tree reflects the nested structure of an XML document. To compute the SSim, we use the numerical string that is composed at nodes' level. Kim [3] assumed that two documents are structurally similar when they are structurally identical or structurally contain in each other. Therefore, after transferring tree to string, a matching algorithm is used to measure the similarity. In this paper,

we calculate similarity by using approximate matching. Accordingly, we assess the structure similarity with regard to common edges of two trees. First, NS is built for *d1* and *d2*. Then, the bigrams of the two strings are calculated. The reason that bigrams are used is to preserve the nested structure of the XML document. Finally, we compute the SSim through (2).

B. Content Similarity (CSim)

Content similarity is obtained by using ES and equation (2). When we want to specify content similarity (CSim) for an XML document, we should consider the semantics of nodes besides an equality between them. As mentioned before, the weight string can represent the position of nodes according to the root; this can be considered as a weight for each node. Therefore, we can make a fuzzy set the members of which belong to name string, while the fuzzy memberships are obtained by the weight string. To compute the CSim we make a fuzzy set for each one of the two XML documents. Then we specify CSim by using equation (1) and fuzzy intersection. In this paper, we use Zadeh 's intersection [15]. In equality of nodes' name, we use WordNet [16] to increase accuracy. Equality can be based on name or value of element, or on both of them. Introducing a coefficient λ, the equality is computed as follows:

$$Eql(S_1,S_2) = \lambda * Ename(S_1,S_2) + (1-\lambda) * Evalue(S_1,S_2), (4)$$

where Ename and Evalue are two functions that compute the equality according to the name and value respectively. Coefficient λ is adjusted by the user, according to the importance assumed for each function.

The time complexity of DFS is O(*n*), where the number of nodes of a tree is *n*. the time complexity of pattern matching is O(*n*+*m*) when the length of a text is *n*, and the length of a pattern is *m* [17]. When the number of nodes of a tree is *n*, the time complexity to obtain fuzzy similarity between two XML documents is O

(2*n*). Therefore, the time complexity of the algorithm is linear in the combined size of the two XML documents involved in the evaluation.

VII. EVALUATION

In our experiments, we used the dataset of Lotus Hill Research Institute (LHI) [18]. This dataset includes 1767 images that have classified into eight categories and 150 subcategories. In addition, each image has an annotation in XML format.

To evaluate our algorithm, we computed the similarity of XML files in identical and different categories.

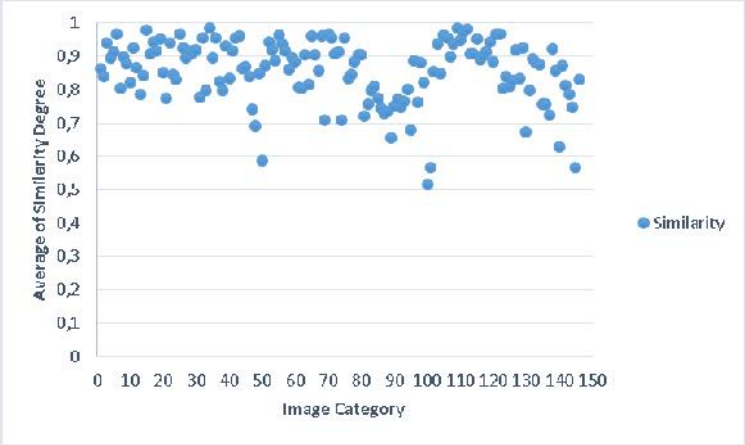

Figure 4. Similarity of images annotated in the same category

Fig. 4 shows the average similarity between the images annotated in the same categories. Clearly, the images placed within a category should have high similarity, which is the result properly obtained in Fig. 4.

We also experimented to determine the similarity between two XML documents in different categories. As is shown by the results of this comparison (Fig. 5), the similarity degree is mostly less than 0.5 as expected. Some cases that have a degree higher than 0.5 belong to situations where two categories are quite similar. For example, subcategory Street is in both Object and Segmentation categories.

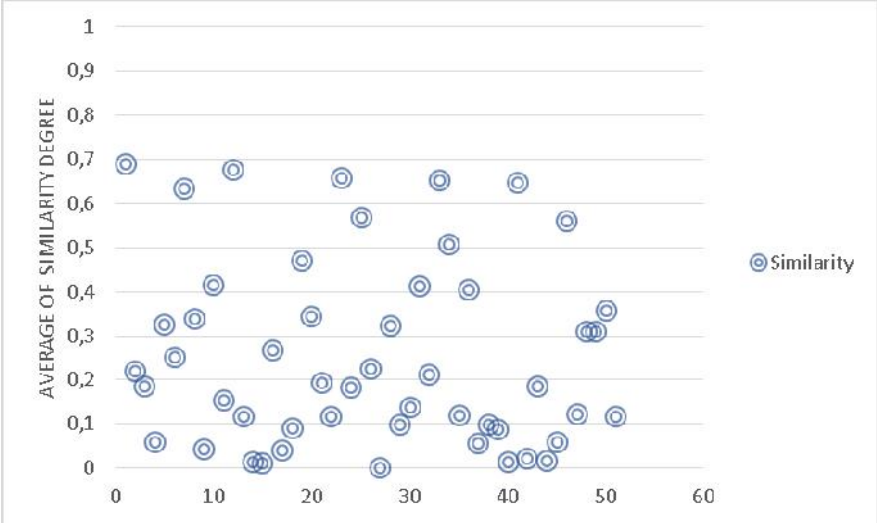

Figure 5. Computing the similarity of images annotated in different categories

VIII. CONCLUSION

We have formalized the problem of measuring similarity between two XML documents. The presented algorithm computes similarity by considering content and structure of XML documents. To compare two XML documents, we have used the strings made according to the ordered labeled trees of them. To increase the accuracy, we have used fuzzy metrics. The time complexity of the algorithm is linear w.r.t. the combined size of the two XML documents. Our future research includes the development of a clustering model based on the proposed approach and the experimental verification of the model.

REFERENCES

[1] A. Aïtelhadj, M. Boughanem, M. Mezghiche, and F. Souam, "Using structural similarity for clustering XML documents," in *Knowledge and Information Systems*, 2012, vol. 32, no. 1, pp. 109–139.

[2] K. L., S. W., and C. K.J., "Semantic Mapping of XML tags using Inductive Machine Learning," *International Conference on Information and knowledge Management*, 2002.

[3] W. Kim, "XML document similarity measure in terms of the structure and contents," *COMPUTER ENGINEERING and APPLICATIONS (CEA)*, pp. 205–212, 2008.

[4] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol. 337, no. 1–3, pp. 217–239, Jun. 2005.

[5] U. Park and Y. Seo, "An Implementation of XML Documents Search System based on Similarity in Structure and Semantics," *International Workshop on Challenges in Web Information Retrieval and Integration(WIRI)*, pp. 97–103, 2005.

[6] G. Li, X. Liu, J. Feng, and L. Zhou, "Efficient Similarity Search for Tree-Structured," *SSDBM*, pp. 131–149, 2008.

[7] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, 2001.

[8] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," *Theoretical Computer Science*, vol. 92, no. 1, pp. 191–211, Jan. 1992.

[9] L. Gravano, P. Ipeirotis, and H. Jagadish, "Approximate string joins in a database (almost) for free," *VLDB*, 2001.

[10] R. Behrens, "A grammar based model for XML schema integration," *British National Conferenceon Databases (BNCOD)*, pp. 172–190, 2000.

[11] H. Prüfer, "Neuer beweis eines satzes uber permutationen," *Archiv fur Mathematik und Physik*, vol. 27, pp. 142–144, 1918.

[12] C. Li, B. Wang, and X. Yang, "VGRAM: Improving performance of approximate queries on string collections using variable-length grams," *VLDB*, 2007.

[13] C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," *ICDE*, 2008.

[14] G. Kondrak, A. Hall, C. Tg, D. Marcu, K. Knight, and M. Rey, "Cognates Can Improve Statistical Translation Models," *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics(HLT-NAACL)*, pp. 46–48, 2003.

[15] L. Zadeh, "fuzzy set," *Information and control*, 1965.

[16] G. a. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995.

[17] D. Knuth, J. Morris, Jr, and V. Pratt, "FAST PATTERN MATCHING IN STRINGS," *SIAM journal on computing*, vol. 6, no. 2, pp. 323–350, 1977.

[18] B. Yao, X. Yang, and S. Zhu, "Introduction to a large-scale general purpose ground truth database: methodology, annotation tool and benchmarks," *Energy Minimization Methods in Computer Vision (EMMCVPR)*, pp. 169–183, 2007.