A Text-based Similarity Join for XML Data

Lianzi WEN†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering
†† Center for Computational Sciences
University of Tsukuba, Japan

Abstract This paper proposes a text-based similarity join method for XML data based on XML data serialization. With the recent explosive diffusion of XML, great volumes of electronic data are now marked up with XML. As a consequence, a growing amount of XML data conveys similar contents, but each may have its own extra information not found in others. To extract as much information as possible from this heterogeneous information, similarity join has been used. In our previous work, we proposed a similarity join scheme that is appropriate for data-oriented XML with regular structures, while it is sometimes inappropriate for such XML data with irregular structures. To cope with this problem we propose a new method which can be summarized as follows: 1) We scan two given XML data to extract XML subtrees by giving a range of the number of text nodes; 2) We filter out dissimilar pairs using similarity of textual information; and 3) We extract pairs as the final result by checking similarity of structural information. We conduct experiments to evaluate our new method by using both real bibliography and bioinformatics XML data. The experimental results show that our method can effectively solve the problem of previous work for both bibliography and bioinformatics XML data, and hence improve the precision of the previous algorithm.

Key words XML, similarity join, XML data serialization, bloom filter

1. Introduction

XML (Extensible Markup Language) [12] is a general-purpose markup language for data exchange on the Internet and has been recommended by the World Wide Web Consortium (W3C) since 1998. XML can represent contents and structure of data through a combination of nested tags; it can be coded hierarchically as a tree structure. XML is currently used in many applications, including web data, business data, science data, logs, etc. It is popular because it can represent any kind of data from multiple sources. As a consequence, the use of XML is expected to continue growing enormously.

As XML becomes increasingly popular for data representation, a growing amount of XML data contains similar contents. For example, DBLP Bibliography and ACM SIGMOD are published in XML on the Internet. The two XML data convey similar contents, but each may contain its own extra information which is not found in the other. To extract as much information as possible from this heterogeneous information, we must efficiently measure similarity between XML data for integrating such similar data sources. Figures 1 and 2 show an example of two XML subdocuments from different databases. Figure 1 has additional information about the publisher and Figure 2 has web side information. Figure 1 represents author information by the authors tag; Figure 2 shows the author name in more detail. In Figure 1, the publication year is expressed by the year element, but in Figure 2, it is expressed by the year attribute. Thus, it is important to integrate such data sources so that users can conveniently access and acquire more complete and useful information.

The operation used for this purpose is similarity join. So far, similarity join has been actively studied in relational databases, and research applied to XML data is underway. The importance of similarity join is expected to increase in the future as XML data continues to proliferate.

However, several problems arise in XML similarity join. The tasks of extracting XML subtrees from XML data at a proper position and measuring the approximate similarity between XML subtrees are not easy to perform. Not only differences in markup vocabulary, but also the order in which elements appear and the tree structure of the XML data might differ. Thus, we have to accurately measure the approximate similarity between XML data, even if there are heterogeneities in content and structure. If the data is similar, similarity join processing should be possible.

In our previous work [15], we proposed an approach for XML similarity join using tree serialization. In general, it
is extremely time-consuming to measure similarity between distinct XML data due to their tree structure representation of XML data. To address this problem, we used serialization of XML tree, which is the process of converting an XML tree into node sequences by traversing the tree in a particular order, thereby making it possible to measure the similarity of XML trees as sequences. The outline is as follows (see [15] for more details):

(1) We extract semantically/structurally coherent XML subtrees. This process can be executed automatically by giving some parameters.

(2) We output node sequences that correspond to the extracted XML subtrees. In this serialization, we use postorder based on NoK (Next-of-Kin) [17] pattern.

(3) We measure similarity of the textual information and filter out dissimilar sequences. In this measurement, we use Bloom filter for speed and effectiveness.

(4) We extract pairs of sequences as the final result by checking similarity of structural information.

By using this algorithm, we conducted several experiments and showed the previous scheme is reasonable in finding similar pairs from bibliography XML data. However, there is a problem in the previous work. Specifically, we intended to extract fixed size of XML subtrees specified by some parameters. This is appropriate for data-oriented XML with regular structures, such as repeated occurrences of similar elements, while it is sometimes inappropriate for such XML data with irregular structures.

In this paper we attempt to use a novel method to extract XML subtrees for subsequent XML similarity join. In the extraction process, we output an element node if it has a number of text nodes as its descendants within the predefined range. Another distinction from our previous method is that we permit overlaps among XML subtrees. We conduct experiments using both real bibliography and bioinformatics XML data to show that our new method is usable for both bibliography and bioinformatics XML data, thereby improving the similarity join algorithm.

The rest of this paper is organized as follows: Section 2 briefly describes related work. Section 3 discusses the proposed extraction method with overview of our previous algorithm which is finding similar pairs by considering text information and structural information. Effectiveness is evaluated in Section 4 and Section 5 concludes the paper with an outline of future work.

2. Related Work

Liang et al. [6], [7] proposed a scheme of approximate XML join. In their approach, they also mentioned a cluster (extraction) method of independent subtrees for approximate XML join. However, their extraction method cannot deal with XML data having irregular structures and it misses some subtrees which are supposed to be extracted. They measure approximate similarity between two subtrees by calculating the similarity of text contents and path expressions. If the similarity is higher than a given threshold, the two XML subtrees are judged similar. When calculating similarity, only identical text nodes are considered and the number of shared tag names are counted over two distinct path expressions.

Weis et al. [14] proposed an approach to detect duplicate objects in an XML data for data cleaning. An object in their paper is an independent XML subtree. In their approach, every element is compared by adopting a top-down traversal of its hierarchical structure. Although their measure for object similarity achieves high precision in detecting duplicates, their approach takes a long time to compare individual pairs of objects and consumes a great amount of memory.

It is well known that tree edit distance (TED) [16] is a metric for measuring the structural similarity in XML data. It is defined as the minimum cost operations (insertions, deletions and substitutions) required to transform one tree to another. It is difficult for TED to be applied to massive data, because
3. The Proposed Scheme

Figure 3 shows an overview of the proposed scheme. In our previous work, we scan two given XML trees being processed to extract XML subtrees. The extracted subtrees are serialized to node sequences for comparing textual and structural similarity.

3.1 Extraction of XML subtrees

The first part of our scheme is subtree extraction from XML data. We observe that our previous extraction method is good at dealing with data oriented XML documents, which have relatively regular structure. However, if a document has a flexible structure, our method is not always successful in extracting subtrees. For example, let us take a look at such examples. uniprot_sprot.xml [10] and uniprot_trembl.xml [11] are XML data in bioinformatics area. Figure 4 shows an example of XML tree structure from uniprot_sprot.xml. Our previous method cannot be successful in extracting subtrees appropriately.

To address this problem, we propose a novel subtree extraction method. We give a range of number of text node which is \( r_{\text{min}} \) and \( r_{\text{max}} \), where \( r_{\text{min}} \) is the minimum number of text nodes and \( r_{\text{max}} \) is the maximum number of the text nodes. We can extract XML subtrees if they have a number of text nodes that fall in the given range. We also allow a subtree to have overlaps with other subtrees. In this way, we can extract many subtrees from XML data.

To extract XML subtrees, we use SAX to traverse each XML element node and count the number of text node occurrences in the subtree rooted at the element node. When we find an element node that satisfies the predefined threshold, we extract that subtree and output each sequence of structural and textual information as we see in Figure 3.

Since the extraction process permits overlapping among distinct (extracted) subtrees, some nodes may be included in some subtrees as long as they satisfy the given condition. By this method, we try to extract many variations of XML subtrees for the text node, and it is considered to be appropriate for XML data with complex structure. In Figure 4, we can extract subtrees such as rooted at gene, lineage, organism, authorList, citation and so on if their number of text nodes are fall in the given range even some subtrees are overlapped such as these subtrees that rooted at lineage and organism.

3.2 Serialization of XML data

In order to measure similarity of extracted XML subtrees, we serialize XML subtrees in node sequences in a tree-traversal order (e.g., pre-and postorder). Postorder is used in ViST [13] and Prüfer order is used in PRIX [9]. In general, information on structure is lost by serializing the tree structure to node sequences (1). In Figure 5, the postorders are the same as “ bca ” although the two XML tree structures differ. Therefore, we see that structural information is lost.

To cope with this problem, the NoK pattern [17] is used in this serialization to maintain structural information. This paper uses postorder instead of the preorder used in the original NoK pattern. One way to materialize the tree is to store the nodes in postorder and maintain the tree structure by properly inserting pairs of parentheses as introduced in [5]. For example, “((b)(c)a)” is a string representation of the tree (Figure 5). The “(” preceding a indicates the beginning of a subtree rooted at “a”; its corresponding “)” indicates the end of the subtree. In the NoK pattern, it removes all closing parentheses and retains only open parentheses as in “((b|ca)” because each node (a character in the string) actually implies a closing parenthesis. Even if a closing parenthesis is omitted, tree structural uniqueness is kept as we see in Figure 5.

3.3 Similarity Measure

In the relational model, data is stored in flat tables where data is simply defined as a single row of such a relational
Table. All data in the table have the same structure and thus only vary in content. In the world of XML, there are not necessarily uniform and clearly defined structures like tables. Two XML data thus have to be compared according to both data (text nodes and attribute values) and structures (elements nodes). Therefore, our technique to detect similar sequences that correspond to an XML subtree has two steps. We measure similarity of textual information first and then check similarity of structural information only for the pair of sequences having similar textual information.

3.3.1 Comparison of textual information

Textual information in this work is defined as a set of words included in text nodes or attribute nodes being considered; therefore, we measure similarity of the text nodes and attribute nodes in which the comparison is being attempted. For example, suppose we want to calculate two pieces of textual information \( t_1 = \text{“database systems Hector Jennifer”} \) and \( t_2 = \text{“database systems Jennifer Jeffrey”} \). In this work, we use Jaccard similarity to calculate the ratio of words common to both \( t_1 \) and \( t_2 \) among all words in \( t_1 \) and \( t_2 \). Jaccard similarity between \( t_1 \) and \( t_2 \), \( JR(t_1, t_2) \), is defined as

\[
JR(t_1, t_2) = \frac{wt(t_1 \cap t_2)}{wt(t_1 \cup t_2)}
\]

From this formula, the similarity of textual information \( JR(t_1, t_2) = 0.6 \). We can say that the textual information is similar if the calculated similarity is above the given threshold \( \alpha \). Notice that it is quite time consuming if we compute Jaccard similarity for all possible textual information over the extracted subtrees.

To cope with the problem, we use the idea of the Bloom filter [3] to accelerate the processing. Bloom filter provides a probabilistic way to determine if an element is a member of a given set. An empty Bloom filter is a bit array of \( m \) bits, all set to 0. There must also be \( k \) different hash functions defined, each of which maps a key value to one of the \( m \) array positions. Figure 6 provides an example of Bloom filter with \( k \) hash functions. If we insert text nodes “database” and “systems” which are included in an extracted subtree, into the Bloom filter, the bits of \( h_1(database) \), \( h_1(systems) \), \( h_2(database) \), \( h_2(systems) \), \( \cdots \), \( h_k(database), h_k(systems) \) are set to 1 in the bit array.

When we calculate similarity of textual information, we first transfer text nodes of the every subtree into a bit array. An important point to note here is that we can estimate the similarity of textual information by calculating the similarity of bit arrays. We can calculate the similarity of bit arrays with the following formula:

\[
Sim(t_1.sig, t_2.sig) = \frac{\text{count}(t_1.sig AND t_2.sig)}{\text{count}(t_1.sig OR t_2.sig)}
\]

where \( t_1.sig \) and \( t_2.sig \) denote bit arrays that are hashed from textual information of subtrees \( t_1 \) and \( t_2 \), respectively, and \( \text{count()} \) is a function that counts the number of “1” in the bit array. Because the similarity of the bit arrays is guaranteed to be larger than the real value,

\[
Sim(t_1.sig, t_2.sig) \geq JR(t_1, t_2)
\]

We can use it to filter out unnecessary candidates. Finally, we refine the resulting candidates by computing real Jaccard similarities to get rid of false positives.

3.3.2 Comparison of structural information

We must check structural information of the pairs having textual similarity. Structural information here is a sequence of the element nodes based on NoK pattern. If the sequences
of element nodes are similar, it can be said that structural information is also similar, since the sequences made by postorder NoK pattern maintains the tree structure. Therefore, we calculate similarity of sequences of element nodes for structural information.

For this measure, we use edit similarity, which is based on edit distance. Edit distance is a similarity measure between two strings. It is defined as the minimum number of point mutations required to change one string to another, where a point mutation is substitution, insertion, or deletion of a letter. Several variations are possible, depending on how point mutations are defined and weighted. In this work, to maintain brevity, we use Levenshtein distance, where substitution, insertion, and deletion are permitted and are equally weighted.

From the definition, for a given couple of strings, the edit distance may vary depending on their lengths. Such a situation is undesirable when we attempt to use it as a similarity measure. In order to cope with this problem, we use edit similarity [2] in which edit distance is normalized by the lengths of strings being compared. Edit similarity can be obtained from edit distance by the following formula:

$$ES(\sigma_1, \sigma_2) = 1 - \frac{ED(\sigma_1, \sigma_2)}{\max(\sigma_1, \sigma_2)}$$

where $\sigma_1$ and $\sigma_2$ denote strings being compared, and $ED$ and $ES$ denote edit distance and edit similarity, respectively.

When comparing a given two XML subtrees, we need only to regard each location step (tag name) as an alphabet. Suppose that we attempt to compute the edit similarity between the NoK pattern based sequences $d_1 = "< ( e ( ( f ( g ( k b a ) "$ and $d_2 = "< ( d ( e c ( ( f ( g b a ) "$ respectively. The edit distance can be computed as follows:

<table>
<thead>
<tr>
<th>$d_1$</th>
<th>$d_2$</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>()</td>
<td>0000011101</td>
</tr>
<tr>
<td>()</td>
<td>( )</td>
<td>00000110</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
<td>0000011</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
<td>000001</td>
</tr>
<tr>
<td>( )</td>
<td>( )</td>
<td>00000</td>
</tr>
</tbody>
</table>

From the result that $ED(d_1, d_2) = 6$, we have $ES(d_1, d_2) = 1.0 - \frac{6}{2} = 0.57$. Finally, we judge that the two pieces of textual information are similar if the calculated similarity is above the given threshold $\beta$.

In this edit similarity measure, we can use filtering technique which is proposed in our previous work [1], for faster computation if there are many pairs to calculate their edit similarities.

3.4 Detecting overlapping pairs

In the results of similar pairs whose textual and structural similarities are above the given thresholds, there exists some overlapped pairs because we permit overlapped subtrees in subtree extraction. We call such a pair “overlapping pair”. We think that overlapping pairs should be avoided as much as possible, because it conveys redundant information.

For an overlapping pair, there are two possibilities to deal with them:

1) Output all resulted subtrees if their similarities are above the given thresholds even there is overlapping pairs.

2) Detect overlapping pairs and delete them from the results. This case gives user only unique information for similarity join.

The choice is up to the user who is doing the process. It should be noticed that it is costly to detect overlapping pairs when a subtree is similar with many nested ancestor and descendant subtrees. For the first case, we just output the results that is obtained from our algorithms. For the second case, we use an illustrative example to explain how we can get rid of redundant pairs. For example, suppose that we have similar pairs as follows: $P_1(A_1, B_1), P_2(A_1, B_{11}), P_3(A_1, B_{12}), P_4(A_2, B_{23})$ and $P_5(A_6, B_{55})$. Here, $A_n$ denote subtrees extracted from XML data $A$ (base XML data). $B_n$ denote subtrees extracted from XML data $B$ (target XML data). For detecting overlapping pairs, first we group the pairs according to the subtrees from base XML data ($A_n$):

1. Group 1: $\{ P_1(A_1, B_1), P_2(A_1, B_{11}), P_3(A_1, B_{12}) \}$
2. Group 2: $\{ P_4(A_2, B_{23}) \}$
3. Group 3: $\{ P_5(A_6, B_{55}) \}$

Then for each group, we decide that two pairs are overlapping if distinct $B_n$ have ancestor-descendant relationship, and we try to omit dissimilar pairs. Taking Group 1: $\{ P_1(A_1, B_1), P_2(A_1, B_{11}), P_3(A_1, B_{12}) \}$ for example, $B_1$ is an ancestor of $B_{11}$ and $B_{12}$. We compute the textual similarity ($sim$) between $A_1$ and $B_{11,12}$. If $sim(A_1, B_1) > max(sim(A_1,B_{11}), sim(A_1,B_{12}))$, then we omit $B_{11,12}$, because $B_1$ subsumes the rest and it is the most similar subtree to $A_1$. Otherwise, we omit $B_1$ and take $B_{11,12}$.

4. Experimental Evaluation

We used a 2-way Dual Core AMD Opteron(TM) processor (2.4GHz) with 16GB memory running Sun OS 5.10. The program was implemented using J2SE 1.5, and we used PostgreSQL 8.1.0 as the underlying RDBMS.


4.1 Extraction of XML subtrees

Table 1 shows the number of independent subtrees extracted from each XML data by new extraction method. Because the original protein XML data are too big to be used in our experiments, we extract only 10000 subtrees from
Table 1 Extraction of XML subtrees.

<table>
<thead>
<tr>
<th>XML data</th>
<th>Size (KB)</th>
<th># of subtrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGMODRecord.xml.xml</td>
<td>464</td>
<td>8,781</td>
</tr>
<tr>
<td>DBLP.xml</td>
<td>400,170</td>
<td>983,188</td>
</tr>
<tr>
<td>uniprot_sprot</td>
<td>938</td>
<td>10,000</td>
</tr>
<tr>
<td>uniprot_trembl.xml</td>
<td>1038</td>
<td>10,000</td>
</tr>
</tbody>
</table>

uniprot_sprot.xml and uniprot_trembl.xml respectively.

In our experiment, we extracted XML subtrees by setting an appropriate range (two parameters $r_{\text{min}} = 3$, $r_{\text{max}}=30$) by taking into account the features of XML data.

Figure 7 illustrates many resulted subtrees which are extracted from uniprot_trembl.xml by the new extraction method. In that Figure, subtrees closed with boxes are extracted subtrees and we labeled a number for each extracted subtree in extraction order. Subtree 2 is descendant of subtree 3, and subtree 4 and 5 are descendant of subtree 6 and so on, but each of them is can be said as an independent subtree. From the results, we can say that our method can extract many variations of subtrees even for XML data with complex structure.

4.2 Textual similarity computation

We tested the effectiveness of the proposed textual similarity computation method. We measured elapsed time for processing 1,384,239,960 ($= 1,530 \times 904,732$) pairs from SIGMOD Record.xml and DBLP.xml respectively, with different textual similarity thresholds. For Bloom filter, we used three hash functions of 256 bits. Table 2 shows 1) number of pairs remaining after Bloom filter, 2) number of correct pairs according to Jaccard similarity, and 3) elapsed time for the two processes. From the result, we observe that the method is successful in filtering dissimilar pairs, and the selectivity of Bloom filter is more than 99% on average.

We additionally investigated the efficiency of the scheme, in particular to see the benefit of Bloom filter, by comparing processing time of the proposed method and that without the filtering step by Bloom filter. That is, for the baseline, we attempted to compute Jaccard similarity for each pair. Figure 8 illustrates the elapsed time. The similarity threshold is 0.5. We observe that the baseline is much slower than the proposed scheme, thereby showing the effectiveness of the filtering step by Bloom filter.

4.3 Structural similarity computation

To evaluate the efficiency of our algorithm based on XML serialization, we compared the processing time with that of the tree edit distance (TED) for 10,000 ($= 100 \times 100$), 50,000 ($= 100 \times 500$), 100,000 ($= 100 \times 1000$), and 1,000,000 ($= 1,000 \times 1,000$) pairs from SIGMODRecord.xml and
DBLP.xml. Figure 9 shows the result. The figure shows that our approach is much faster than TED, which is known to be costly to process. With our approach, we can achieve good performance even for large XML datasets.

4.4 Extracted pairs

Figures A·1 and A·2 show results of finding similar pair of subdocuments from bibliography data. Figures A·3 and A·4 show results from bioinformatics data. As we can see, these resulted pairs represent similar information. Therefore, we can say that the proposed scheme is successful in finding similar pairs from both bibliography and bioinformatics XML data and it do not depend on XML size and their structures.

5. Conclusions

In this paper, we proposed an improved algorithm for similarity join. We proposed new extraction method and our experimental results suggest that the proposed method is appropriate for XML data with complex structure. In the similarity measure, we take into account both textual and structural information. Experimental results suggest that the proposed scheme is reasonable in both accuracy and effectiveness.

In the future, we plan to do further experiments to evaluate our scheme using different types of large XML data.

Acknowledgments

This research has been supported in part by the Grant-in-Aid for Scientific Research of JSPS (#18560018 and #1970083) and of MEXT (#19024006).

References


Appendix
Figure A-1 Extracted similar subdocuments (1).

Figure A-2 Extracted similar subdocuments (2).

Figure A-3 Extracted similar subdocuments (3).

Figure A-4 Extracted similar subdocuments (4).